INTEGRATION OF CELLULAR ASSISTED HETEROGENEOUS NETWORKING AND BLUETOOTH SERVICE DISCOVERY PROTOCOL

Diplomarbeit

der Philosophisch-naturwissenschaftlichen Fakultät

der Universität Bern

vorgelegt von

Simon Winiker

2004

Leiter der Arbeit:

Prof. Dr. Torsten Braun

Forschungsgruppe Rechnernetze und verteilte Systeme (RVS)

Institut für Informatik und angewandte Mathematik

Betreuer der Arbeit:

Marc Danzeisen

Forschungsgruppe Rechnernetze und verteilte Systeme (RVS)

Institut für Informatik und angewandte Mathematik

ACKNOWLEDGEMENTS

First of all I want to thank my family for their support and infinite confidence in me. Further my thanks go to my girl-friend Claudine for the encouraging words during the whole time I was working on my diploma.

Thanks to Swisscom AG, who provided the material and the resources needed for my diploma work. Further I would like to thank all the people at Swisscom Innovations for their cooperation. It was an inspiring time working with you and I collected very valuable experiences.

My special thanks go to Prof. Dr. T. Braun for making this cooperation with Swisscom AG possible and for care during my diploma work. Special thanks also to M. Danzeisen and Dr. D. Rodellar for the technical assistance and support during my diploma work.

Further I would like to thank J. Linder for all his help during my time in Swisscom AG, M. Schädler for the provisioning of the needed hardware and E. Maghsoudi for his patient assistance during the test phase of this diploma work.

Last but not least I want to thank all my friends. They will know for what.

ABSTRACT

New computer applications and the development of user end devices show a clear trend towards mobility. Devices become smaller and thus portable or even wearable. So-called road warriors, employees who work while being on the move, need to access their enterprise data, while they are out of the office.

This need for mobility is generating troubles in terms of network organization and security, as this data is often quite sensitive. Technologies like MobileIP [1] try to cope with the mobility problem on the behalf of the network layer and Virtual Private Networks (VPN, a good overview is presented in [2]) are dealing with security issues for remote intranet accesses. SecMIP [3] is a solution, which mixes up these mechanisms to provide a secured MobileIP mechanism.

Additionally there is not just one access technology for mobile devices, what causes access problems. Access Technologies like Bluetooth [4], WLAN [5] and GSM [6] are different, and so the user has to have the right interface with the right configuration to access a certain network. Coping with these configurations can be quite a hassle for the inexperienced user and applications become available to ease this task.

After all, there are also applications, which are not prepared for mobility. For example a printer can not just be omnipresent to offer the mobile user its service. As a solution for this kind of problems, public services, like public printers, become available. Again, it can be really complicated for a user to access such public services. Service discovery protocols cope with those issues and define a set of protocols for dynamic server-client applications. Dynamic means that a service provider can share a certain service, which a client can search for and access with help of the protocols provided by a service discovery protocol Environment.

Deeper analysis of service discovery protocol specifications shows two main issues for a possible deployment: the lack of security and the need for Multicast. Security is important for today's applications as sensitive data between the service providers and its users has to be protected. To prove the authenticity of messages exchanged between the players in a service discovery protocol environment, authentication mechanisms have to be implemented. Authentication is also crucial for possible billing infrastructures. On the other hand support for Multicast in large scale networks is improbable which has a negative impact on the scalability of service discovery protocol Environments. Other means for the transport and the spreading of messages has to be found to transport messages to multiple users.

CAHN (Cellular Assisted Heterogeneous Networking) is an approach for managing devices in a heterogeneous environment, which can be the result of the mentioned trend towards mobility. In this approach it is suggested to separate the signaling plane from the data plane for the establishment of an automated and secured data link between the users. For the signaling plane a cellular network is suggested, which qualifies itself by a high coverage. The existing trust relation between the user and the operator of a cellular network can build the basis for authentication. These features make CAHN interesting in terms of the mentioned problems of Service Discovery Protocols.

This diploma now aims at providing an integration of the CAHN approach and Service Discovery Protocols. To do so the Bluetooth Service Discovery Protocol builds the basis for the development of an application integrating the two technologies.

This diploma work shows, that the integration of the two technologies is very promising and both technologies can profit from the integration. Bluetooth will gain an improved authentication method and scalability and CAHN can indicate its features via the Bluetooth Service Discovery Protocol. But it will also become clear, that further work in the topic is necessary to improve the behavior of the provided application. Further this diploma work can not treat the mentioned Multicast issue, certain Service Discovery Protocols have. The idea that Multicast messages could also be sent over the signaling plane provided by CAHN has not been analyzed so that no conclusions can be drawn at the moment and this makes this issue topic of future work.

TABLE OF CONTENTS

1 INTRODUCTION 1

1.1 DIPLOMA WORK DESCRIPTION 1
1.2 DOCUMENT STRUCTURE 3

2 BLUETOOTH BASICS 4

2.1 GENERAL 4
2.2 THE BLUETOOTH PROTOCOL STACK 4
2.3 PICONETS AND SCATTERNETS 6
2.4 INQUIRY AND PAGING 6
2.5 SERVICE DISCOVERY 7
2.6 BLUETOOTH PROFILES 7
2.7 BLUETOOTH SECURITY 8
2.8 THE BLUETOOTH NETWORK ACCESS POINT SCENARIO 9
2.9 SECURITY LIMITATIONS OF THE PAN PROFILE 9

3 SERVICE DISCOVERY PROTOCOLS 11

3.1 INTRODUCTION 11
3.2 JINI 12
3.2.1 CONCEPTS 12
3.2.2 COMPONENTS 14
3.2.3 SERVICE ARCHITECTURE 15
3.2.4 THE PROTOCOL STACK 16
3.2.5 SELF CONFIGURATION 16
3.3 UPNP 17
3.3.1 CONCEPTS 17
3.3.2 COMPONENTS 18
3.3.3 THE UPNP PROTOCOL STACK 19
3.4 SSDP 22
3.4.1 CONCEPTS 22
3.4.2 COMPONENTS 23
3.5 SALUTATION 25
3.5.1 CONCEPTS 25
3.5.2 COMPONENTS 27
3.5.3 SALUTATION LITE – DOWN-SCALED SALUTATION 28
3.6 BLUETOOTH SERVICE DISCOVERY 29
3.6.1 CONCEPTS 29
3.6.2 COMPONENTS 30
3.6.3 THE PROTOCOL STACK 31
3.7 SERVICE LOCATION PROTOCOL 33
3.7.1 CONCEPTS 33
3.7.2 COMPONENTS 35
3.7.3 RESOURCE DISCOVERY PROTOCOL 36
3.8 CONCLUSIONS 37

4 CELLULAR ASSISTED HETEROGENEOUS NETWORKING (CAHN) 41

4.1 THE BOOTSTRAPPING PROBLEMS 41
4.2 AUTHENTICATION AND SECURED LINK ESTABLISHEMENT BY CELLULAR OPERATORS 42
4.3 THE CAHN TARGET SCENARIO 44
4.4 CONCLUSIONS 45

5 DESIGN 46

5.1 REQUIREMENT SPECIFICATIONS 47
5.2 DEFINITION OF THE CAHN BLUETOOTH SERVICE AND THE CAHN BLUETOOTH SERVICE ATTRIBUTE 48
5.2.1 SERVICES IN BLUETOOTH SDP 48
5.2.2 DEFINITION OF THE CAHN SERVICE 49
5.2.3 DEFINITION OF THE CAHN SERVICE ATTRIBUTE 50
5.2.4 DEFINITION OF A CAHN PROFILE 50
5.3 CAHN PROTOCOL DEFINITION 51
5.3.1 CAHN BLUETOOTH SERVICE REQUEST 51
5.3.2 CAHN BLUETOOTH SERVICE RESPONSE 52
5.3.3 CAHN ERROR RESPONSE 52
5.4 CAHN CORE 54
5.4.1 CAHN COMMUNICATION MANAGER (CCM) 55
5.4.2 CAHN CONNECTOR 56
5.4.3 CAHN ADAPTER 57
5.4.4 CAHN SMS ADAPTER 60
5.5 THE GUI 61
5.6 MESSAGE FLOW 62
5.6.1 CLIENT SIDE 62
5.6.2 SERVER SIDE 63

6 IMPLEMENTATION 64

6.1 INTEGRATION OF THE CAHN BLUETOOTH SERVICE AND THE CAHN BLUETOOTH SERVICE ATTRIBUTE IN THE BLUETOOTH SDP IMPLEMENTATION 64
6.1.1 USED BLUETOOTH IMPLEMENTATION 64
6.1.2 BLUETOOTH SDP MODIFICATIONS 65
6.2 CAHN COMMUNICATION MANAGER (CCM) 69
6.3 CAHN CONNECTOR 72
6.4 CAHN ADAPTER 74
6.5 CELLULAR NETWORK 75
6.5.1 GSMLIB 76
6.6 CAHN SMS ADAPTER 77
6.7 THE GUI 78

7 EVALUATION 90

7.1 THE TEST BED 90
7.2 CONNECTION ESTABLISHMENT TIMES 91
7.3 SMS MESSAGE EXCHANGE DELAYS 94
7.4 BLUETOOTH PAN CONNECTION ESTABLISHMENT TIMES 95
7.5 PERFORMANCE SUMMARY 97

8 CONCLUSIONS 99

8.1 LESSONS LEARNED 99
8.2 FUTURE WORK 100

9 REFERENCES 102

1 INTRODUCTION

1.1 DIPLOMA WORK DESCRIPTION

In the past years the Internet has gained a big importance as a platform not only for non-commercial applications, but also for commercial interests. The client server principle has proved to be very successful for Internet based applications and many companies deploy their own servers for the provisioning of their offers. Online transactions have become a part of the daily life and many clients are using online services. The fast growth of the Internet and the availability of an incredibly high number of services can be confusing for the end user. But not only the access to the services introduces much confusion, but also the configuration and use of the numerous different access technologies, like Bluetooth, Wireless LAN and GSM, can be a hassle for the user. Many services, many logins, many different security mechanisms and much configuration work may be the consequence. Newer technologies moreover enable device mobility, which can permit the spontaneous and ad-hoc formation of networks or communities to share services and resources with other users.

This situation is the key motivation to find a solution, which can provide the needed features for upcoming applications. This includes:

- the dynamic formation and handling of networks without configuration hassles,
- the possibility to share services, browse for shared services and access shared services in a simple way,
- the setup of security relations between the users in the network.

Cellular Assisted Heterogeneous Networking (CAHN) suggests reusing a cellular network to transport the needed configuration and security parameters in a heterogeneous environment, to build up a secured network connection. An environment is heterogeneous, when not all participating nodes share a common access technology. Users in a CAHN environment are identified by their phone number, and can be authenticated by the cellular operator. Therefore CAHN can help to dynamically form and handle a network and can assist the user in the security setup.

Service discovery protocols define sets of protocols, which can be used to share services, search for services and access shared services. Service discovery protocols do have issues, which have to be solved for a successful deployment in heterogeneous and spontaneously built environments, as well as in the Internet.

The suggested approach to deal with the issues of service discovery protocols is to integrate CAHN in the service discovery protocol. This integration will show whether CAHN can help to deploy service discovery protocols in heterogeneous and spontaneously built networks. As CAHN and service discovery protocols both contain interesting aspects for the ease of network formation, service access and security establishment, the integration of the two technologies seems to be a promising approach for the deployment of future services.

This diploma work aims at providing a basic implementation of CAHN and its integration in the service discovery protocol that is defined and used in the Bluetooth technology, further referred to as Bluetooth SDP. This implementation will serve as a proof of concept and will give some practical ideas about the use of CAHN in service discovery protocol controlled infrastructures.

The resulting implementation shows that both, CAHN and the service discovery protocol can profit from the integration. CAHN can be used to integrate security in the service discovery protocol and the service discovery protocol can be used to detect the provided CAHN features of other devices and services in the network. Further the implementation acts on the behalf of the user and thus facilitates the connection and security establishment. With the help of the provided application, the user can easily share services, browse for available services and access shared services in a secure way.

The implementation of the integration is used on a demonstrator to evaluate the practical behavior of the implementation. This shows that the implementation itself performs well, and that the idea of integrating CAHN in the Bluetooth SDP is a valuable approach for enabling secured connections in public Bluetooth environments. On the other hand the evaluation also proves that the implementation of CAHN is too primitive and that especially the SMS mechanism does not suit for the delivery of CAHN protocol messages over the cellular network. Ideas on how this could be improved are presented at the end of this diploma work.

Furthermore it has to be concluded that even if the integration of CAHN in the Bluetooth SDP is a promising approach, it cannot be definitively predicted, whether an integration of CAHN in more sophisticated service discovery protocols is still as valuable or even possible.

1.2 DOCUMENT STRUCTURE

Chapter 1 gives an overview of the motivation for this diploma work and the acquired results. After that the structure of the document is presented.

Chapter 2 introduces the Bluetooth technology. The most important principles of Bluetooth defined in the specifications are explained.

Chapter 3 is about service sharing and access. Service discovery is the approach of interest. The different service discovery protocols share the same basic functionality, but differ in some important aspects, so that they will be presented individually.

Chapter 4 explains CAHN, which is an approach to ease the configuration in heterogeneous environments and helps to establish secured links among the peers.

Chapter 5 introduces the basic idea of merging the CAHN approach with the Bluetooth SDP. Further it identifies the requirement specifications and explains the design of the implementation integrating CAHN and Bluetooth SDP

Chapter 6 contains the explanation of the implementation of the design.

Chapter 7 presents evaluations that were made using the application in a demonstrator.

Chapter 8 concludes the document and presents the lessons that have been learned during this diploma work and suggests future work in the domain.

2 BLUETOOTH BASICS

2.1 GENERAL

The Bluetooth technology is named after the Danish king "Harald Blatand", which united the Scandinavian people during the 10th century. As well, Bluetooth should unite the devices in a Personal Area Network (PAN). The development started in 1998, when Ericsson, IBM, Intel, Nokia and Toshiba formed the Bluetooth Special Interest Group (SIG) to promote a global solution for short-range wireless communication. The goals were primarily to provide a cable replacement with low-power and low-cost chips. To improve the acceptance of the new standard, the Bluetooth SIG decided to offer the specifications to members of the group royalty-free. Today the group has more than 3000 members. In summer 1999 the Bluetooth SIG released the first version of the specifications: 1.0A. By the time of this writing release 1.2 is the most recent version. The Bluetooth specifications are divided in two parts: Part A, Core and Part B, Profiles [4].

2.2 THE BLUETOOTH PROTOCOL STACK

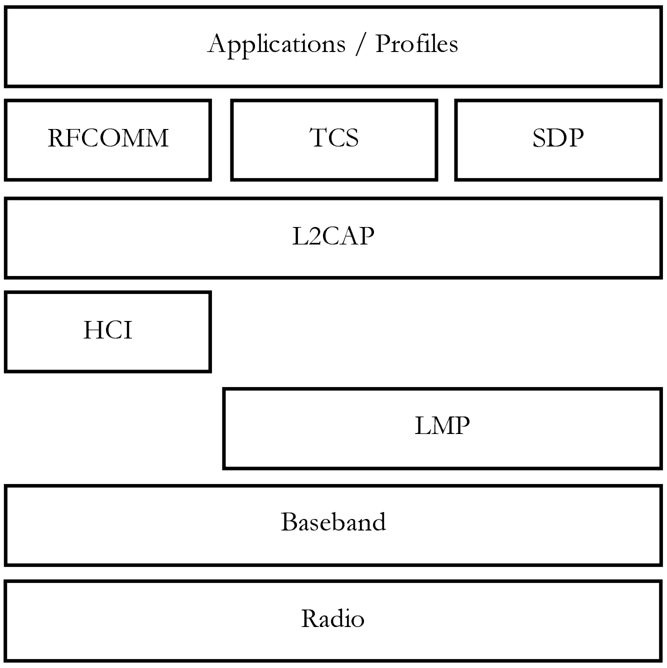


Figure 2.1: The Bluetooth Protocol Stack

In figure 2.1 the schematic building of the Bluetooth protocol architecture defined in the Bluetooth specifications is shown.

The Radio layer defines the technical characteristics of the Bluetooth radio. The Bluetooth radio operates in the 2.4 GHz Band and defines a fast (1600 hops/s) frequency-hopping between 2.402 GHz and 2.480 GHz on 78 channels. Every channel is split in slots with a length of 625 microseconds, which are used for sending and receiving. Further the radio link defines three classes of power consumption for the Bluetooth chip. Class 1 radios use a transmission power of 100 mW, class B radios 2.5 mW and class C radios 1 mW with an estimated range of 100 m for class A radios, 10 m for class B radios and about 1 m for class C radios.

The Bluetooth baseband defines the basic procedures that are used to establish communication between Bluetooth devices, like the formation of a Piconet or Inquiry and Paging mechanisms.

The Link Manager Protocol (LMP) is used by two communicating devices to exchange information about the supported link properties, like the supported power or security modes. Further Bluetooth does define two different link types, the synchronous connection-oriented (SCO) links (i.e. used for voice traffic) and asynchronous connection-less (ACL) links (used for data traffic), which's capabilities my be exchanged using the LMP.

The Host Controller Interface (HCI) provides a standardized interface to the upper layers to control the baseband and the radio. It is used for the connection setup and therefore Inquiry commands, security parameters or packets for connected devices can be passed to the HCI.

The Logical Link Control and Adaptation Protocol (L2CAP) offers a packet interface to the upper layers. The principles of master/client (explained later in "Piconets") do not exist any more on this layer. The L2CAP manages the communication between two devices by multiplexing different logical channels over an ACL link. Further, the L2CAP is responsible for packet fragmentation and reassembling.

The RFCOMM is a standard protocol to simulate a serial (RS-232 conform) link over the air. Upon RFCOMM a PPP link can be established to setup an IP link between Bluetooth devices.

The Telephone Control Signalling (TCS) can be used for binary telephone related controlling of links, i.e. to pick up a call on a Bluetooth headset or to indicate the caller's phone number.

The service discovery protocol and the Profiles will be explained later in this document.

2.3 PICONETS AND SCATTERNETS

A Bluetooth Piconet is formed in an ad-hoc manner and consists of a master device, which creates and controls and up to seven active clients. The master/slave role is a dynamic role and not statically assigned to a device. The master decides which client can send packets at which time. Besides the seven active clients, Bluetooth Piconets can have inactive clients, which are in a so-called parked mode. Devices in parked mode can be set active to join the communication, but only seven clients may be active at the same time. Furthermore, clients can be members of more than one Piconet and what results is referred to as Scatternet. Figure 2.2 shows two Piconets forming a Scatternet.

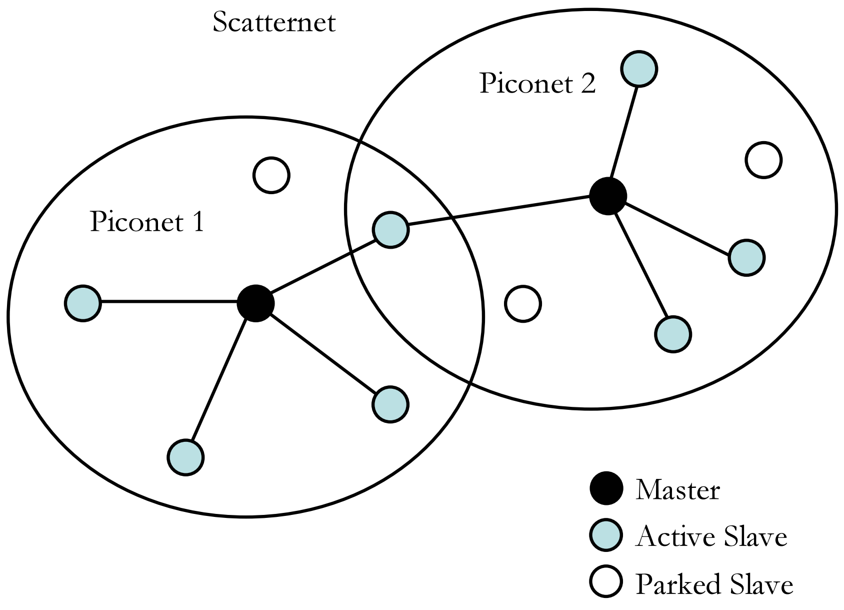


Figure 2.2: Bluetooth Piconets and Scatternet

2.4 INQUIRY AND PAGING

In a Bluetooth environment devices can scan the environment for other Bluetooth devices. For that purpose a message is periodically spread on different frequencies. This frequency altering is important, as Bluetooth uses a frequency hopping and the two devices may therefore miss each other. A device allowing to be discovered enters periodically in the Inquiry scan mode, where it listens to Inquiry messages. If such a message arrives, the device answers with its Bluetooth address and its clock. The clock is needed to synchronize the devices' frequency hopping sequences, if a connection will occur. To prevent other devices from detecting a certain device, this latter device may also be hidden by using the "undiscoverable" option. With this option enabled a device does not enter the Inquiry scan state and can thus not answer to Inquiry messages and will stay undiscovered. Within an Inquiry the discovered device does not get any information about the discovering device. The result of an Inquiry is a list of Bluetooth addresses of possible connection partners with their clocks.

Once the Inquiring device has the knowledge about other devices in its vicinity, it can try to establish connections to them. For that purpose, it does actively page devices to join its Piconet. The initiator of a Paging will become the master of the formed Bluetooth Piconet. If the master already knows a certain device, no prior Inquiries are needed and the Paging can be done immediately.

2.5 SERVICE DISCOVERY

In addition to the feature to discover other devices Bluetooth can also help to discover services supported by other devices. For that purpose Bluetooth does define its own service discovery protocol. More details on this service discovery protocol can be found later in this document. Services can either be detected on a known device to get the list of its capabilities or services may be searched for in the environment. In both cases the Inquiry can be involved to get available devices. In order to be able to announce services and detect services, a common description must be defined in advance. This is achieved by using profiles. Profiles are standardized through the "Bluetooth Special Interest Group" (Bluetooth SIG) and describe the service in detail. Devices enabled with a certain profile can announce this capability and detect other devices with this capability. The use of profiles is fundamental in Bluetooth. In order to protect services and devices from abuse, Bluetooth does define its own security, which will be explained below.

2.6 BLUETOOTH PROFILES

Bluetooth profiles are used to describe services. Profiles are revised and standardized by the Bluetooth SIG. What sets Bluetooth apart from other technologies is that a profile not only describes the involved protocols for a certain service but also possible applications and use cases. With respect to the communication stack, a Bluetooth profile defines the use of the layers for a service. This feature adds value for the end user, as applications are defined, too. All profiles in Bluetooth are depending on the "Generic Access Profile (GAP)" [18], which defines how Bluetooth links can be established and how security settings allow the devices to be detected and the services to be connected. A number of profiles have already been defined and integrated and others are still under development by the Bluetooth SIG. Profiles can also define means how security is used.

2.7 BLUETOOTH SECURITY

As mentioned, devices can be hidden when they are in undiscoverable mode. This does not really enable strong security. In order to protect the services Bluetooth implements its own security model for the access. With help of this model nodes can be authenticated and links can be encrypted. The Bluetooth security model defines three modes of security.

- Security Mode 1: No security procedures are enforced. Unrestricted access.

- Security Mode 2: A device does not initiate security procedures before channel establishment. This mode allows different and flexible access policies for applications, especially running applications with different security requirements in parallel.

- Security modes 3: (link level enforced security): A device initiates security procedures before the link set-up is completed. In this mode it is even not possible to exchange service discovery protocol messages, prior the secure link establishment.

These security modes are defined in the GAP and form the basis for each connection attempt. In order to setup a secured link corresponding to the needed security mode 2 or 3 the accessing device must be authenticated. This authentication is based on a PIN. The PIN furthermore is used to generate a link key, which in turn is the basis for the encryption key, which is re-generated periodically during transmissions. For a secure connection establishment a link key shared by the devices is needed. If no such link key exists, it has to be generated and therefore the user has to provide a PIN. If the PINs entered on both devices correspond the computed link keys will lead to a successful connection establishment. This handshake procedure either happens upon link layer connection establishment or L2CAP connection establishment, depending on the used security mode.

Bluetooth security is representative of the shared secret cryptography. In this approach all participating members have the knowledge about a common key, which is used for both, the encryption and the decryption of the data. Therefore these encryption mechanisms are also referred to as synchronous encryption. The weakness in synchronous encryption approaches is the possible exposure of the shared secret, which makes it possible for an attacker to decrypt all the encrypted data.

In contrast to the shared secret approach, the public/private key cryptography makes use of a key pair to provide data encryption. There the public key is used to encrypt the data and the private key is used to decrypt the data. Therefore these encryption mechanisms are referred to as asynchronous encryption. Every member has to keep secret its proper private key. The public key has to be made available to the members, so they can encrypt messages. If someone's private key is exposed, an attacker can only decrypt messages encrypted with the related public key. Therefore the risk for the community is much lower than in synchronous encryption. Asynchronous encryption is known to perform worse and to be more expensive, than synchronous encryption. Therefore often asynchronous encryption is used to exchange a shared secret to enable synchronous encryption. By using this approach, the shared secret is re-build on every new connection, which makes it hard for attackers to break in.

2.8 THE BLUETOOTH NETWORK ACCESS POINT SCENARIO

The Personal Area Network Profile (PAN profile) defines mechanisms for the use of the Internet Protocol (IP) in a Bluetooth Piconet. For the transport of IP packets over a Bluetooth connection the Bluetooth Network Encapsulation Protocol (BNEP) is used, which is also defined in the PAN profile. Furthermore the profile presents two scenarios, how devices can interact in a Piconet. First the master of a Piconet becomes the Group Network controller (GN) and all the slave devices, also called PAN users (PANU), connect to that GN forming a star topology with the GN as central point. All devices can get assigned an IP address and with help of the BNEP IP-networking is enabled. All messages are routed through the GN. The second scenario differs from the first in the circumstance that the master, in this scenario referred to as Network Access Point (NAP) shares an Internet connection by implementing bridging mechanisms to enable network access to the clients of this Personal Area Network.

With help of this profile it is possible to implement a Bluetooth Access Point. Of course security plays an important role in this scenario and the Bluetooth security modes 2 and 3 can be applied to authenticate the users and encrypt the traffic between the NAP (GN) and the PANUs.

2.9 SECURITY LIMITATIONS OF THE PAN PROFILE

As mentioned above, the Bluetooth security is a shared secret cryptography mechanism. Devices have to agree on a common PIN (key) to enable security. This brings in the known key distribution problem. Users intending to connect to such a Bluetooth network access point must acquire a PIN first. Pre-paid scenarios are imaginable, where a user can buy a ticket containing a PIN. For spontaneous and ad-hoc connection establishment, this requirement can not so easily be fulfilled and automated as the parties must have a common and secured communication channel to exchange the PIN. As the PIN is also needed for authentication, methods like Diffie-Hellman cannot help with this issue. Therefore the Access Point scenario can be easily deployed in small environments, where the users know and trust each other, but the scenario does not scale for public Access Points. Therefore some means for distributing the PINs to the members have to be deployed in a public Access Point scenario.

The targeted integration of CAHN in the Bluetooth SDP will show how the key distribution issue can be solved, and how the Bluetooth PAN profile can be adopted for a public Access Point Scenario, by using the a separate channel to exchange the PIN.

3 SERVICE DISCOVERY PROTOCOLS

3.1 INTRODUCTION

The base strategy to provide a dynamic environment for service sharing and access is used among all the presented service discovery protocols. They mainly consist of three entities: the service catalogue server, the service server and the client. The service catalogue server contains information about the available services in the network. The service server can register information about the service it provides on the service catalogue server. A client can then query the catalogue server for available services and gather information on how to contact the service server and how to access the provided service. Besides these entities, service discovery protocols define also typical actions which can be performed in this environment and the protocol messages, which are exchanged between the different components. These action and protocol definitions differ from one service discovery protocol to the other, depending on the special purpose they pursue. Important actions are for example the discovery of service catalogue servers, the search for services or the registering of services. Bluetooth SDP does integrate the service catalogue server into the service server and is therefore a little different from the other presented service discovery protocols. But as can be seen later, there exist methods to map Bluetooth SDP to more complex environments. Even if the main principle is about the same in the different service discovery protocol proposals, they differ in important aspects, so that the different service discovery protocols are presented individually.

3.2 JINI

A Jini system is a distributed system based on the idea of federating groups of users and the resources required by those users. The goal of the architecture is to turn the web in a flexible and easily administered tool with which resources can be found. The end goals can be summarized as follows:

- Enabling users to share services and resources over a network
- Providing users easy access to resources while network structure changes
- Simplifying the building, maintaining and altering the network

Jini extends the Java system from one single virtual machine to a network of machines. Java is ideal because data and code are portable, because Java provides security and because Java applications can be distributed.

The Jini infrastructure provides mechanisms for devices, services and users to join and detach from a network. The presented specifications are a summary of [7].

3.2.1 CONCEPTS

Services

The service is the most important concept of Jini. A service can be code, hardware, software or even a user. A Jini system should not be thought as a client and server infrastructure, but as a set of services put together to fulfil a certain task. The system provides mechanisms for service construction, service lookup, service communication and service use in a distributed system. Interaction among services is done by the use a protocol. This is a set of interfaces written in Java. This protocol is open ended, but a base protocol is defined within the system.

Lookup Service

The lookup service is used for finding and resolving services. It is the major point between the system and the user. A service is added to the lookup system by a pair of protocols called discovery and join.

Java Remote Method Invocation (RMI)

RMI [8] is rather a part of the infrastructure than a service. It provides mechanisms to find, activate and garbage collect object groups. RMI is a Java extension to normal Remote Procedure Call (RPC) [9] mechanisms. With help of RMI full object code can be transmitted, not only objects. This is one of the key benefits of this system.

Security

The design of the security for Jini is built on twin notions of a principal list and an access control list. Services are accessed on behalf of some entity, the principal, which traces back to a user of the system. Whether access is allowed or not depends on the entries in the access control list. As Java code is moved across the network and executed on the different nodes, the virtual machine security concept known from other Java applications also applies to this code.

Leasing

Access on many services in Jini is lease based. Each lease time is negotiated between the provider and the user of a service as part of the protocol. When the lease is not renewed before it is freed (because the service is not longer needed, not longer allowed or network errors have occurred), the provider or the user can free the lease. Leases are either exclusive or non-exclusive. Exclusive leases ensure that no one else may take a lease on the resource during the period of the lease.

Transactions

A series of operations within a service (or many relied services) is wrapped as a transaction. The Jini transactions supply a service protocol needed to coordinate a two-phase commit. How transactions are implemented is left up to the service using those interfaces.

Events

Distributed events are supported by Jini. An object may allow other objects to register interest in events of the object and receive a notification of the occurrence of such an event. This enables distributed event-based programming with a variety of reliability and scalability guarantees.

3.2.2 COMPONENTS

Jini can be divided in three parts: infrastructure, programming model and services. The infrastructure provides the components needed to build a Jini system. The services are the entities of this Jini system and the programming model is a set of interfaces that enables the construction of services.

Infrastructure

The Jini infrastructure consists of the following three parts:

- Distributed Security: Integrated in RMI, extension to Java security for distributed systems
- Discovery and Join protocols: Allow services to become part of and announce services to the system
- Lookup service: Repository of objects written in Java, that can be downloaded or serve as a proxy

The discovery and join protocols define the way a service becomes part of a Jini system; RMI defines the base language with help of which the services communicate and the distributed security model defines how entities are identified and how they get the rights to perform their actions. The lookup service reflects the current members of the system and acts as the central market place for services.

Programming Model

Services define interfaces through which can be communicated. The sum of these interfaces builds the Jini programming model. There exist three kinds of interfaces:

- Leasing interface
- Event interface
- Transaction interface

The Jini transaction protocol provides two steps to coordinate actions: voting phase and commit phase. The service votes whether and how a task is achieved in the voting phase and in the commit phase the coordinator can commit these changes. The Jini transaction protocol defines the transaction interfaces and transaction objects used to achieve their tasks. The Jini infrastructure now makes use of these interactions. E.g.: The Lookup service uses the lease and event interfaces.

The implementation of services does not need to support the Jini implementation model, but the interfaces of this service.

Service

The Jini technology infrastructure and programming model are built to enable services to be offered and found in the network federation. These services make use of the infrastructure to make calls to each other, to discover each other, and to announce their presence to other services and users. A service has an interface that defines the operations that can be requested of that service. Some of these are intended for the interaction with the infrastructure, others for the user interaction. The type of service defines the interfaces and the methods that can be used to access that service.

3.2.3 SERVICE ARCHITECTURE

Discovery and Lookup Protocols

The heart of the Jini system consists of three protocols, discovery, join and lookup. Discovery and join occur, when a device is plugged in. It first searches for a lookup service, with help of the discovery protocol and when it finds a lookup service, it registers to it with help of the join protocol. An object of the service is then uploaded to the lookup service. Lookup is used when a client or user needs to locate and invoke a service described by its interface type. Then the service can be loaded onto the client.

The service can implement a private protocol between the client and the service provider. The movement of the code from the service provider to the client is one of the most important features of Jini. The client is only dealing with an interface written in Java. Behind this interface RMI may be used to call methods from the server or the methods can be computed locally, or some combination of both, which is called a smart proxy.

A user interface stored on the lookup service offers the possibility to the user to manipulate the service. In fact such a user interface is a special form of a service interface.

If no lookup service is available the client can send a peer lookup to the service provider instead. This one can register his service as he would do with a lookup service. The client can then use this service.

Service Implementation

Objects can share the same address space with other objects especially when there are certain location or security requirements. Such objects make up an object group. An object group is guaranteed to always reside on the same address space or in the same virtual machine.

A service can be implemented directly or indirectly by specialized hardware. Such devices can be contacted by the code associated with the interface for the service.

From a client's point of view, it does not change, whether a service is implemented with objects in the same address space, on the same virtual machine or implemented by hardware. For the client they always occur just as a service which can be used.

3.2.4 THE PROTOCOL STACK

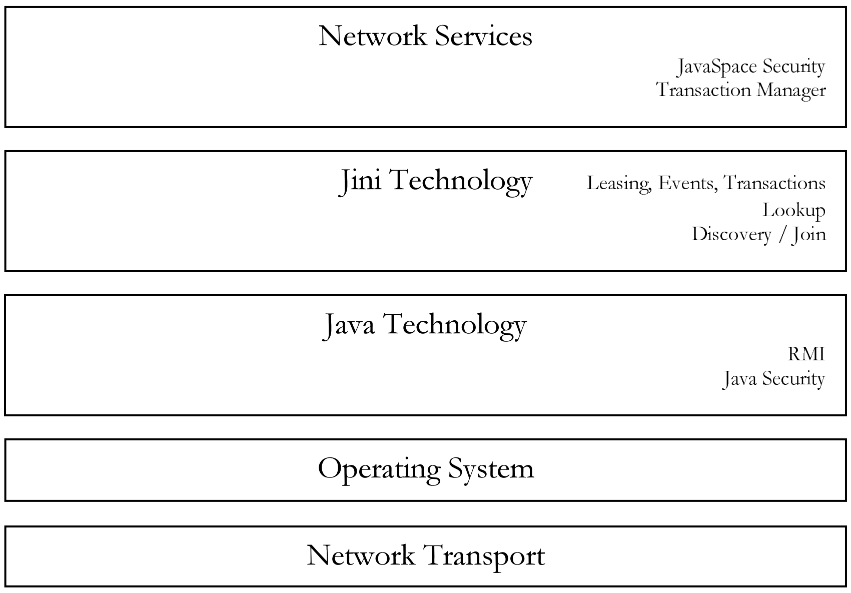


Figure 3.1: The Jini Technology Stack

As can be seen in Figure 3.1 Jini does not define the underlying network structure.

3.2.5 SELF CONFIGURATION

Jini does not directly address this area. An IP device when plugged onto the network will have to be configured with an IP address, a subnet mask and optionally with a gateway and DNS server. From then on, the lookup services can be used. But the use of the JAVA platform means that it cannot directly interfere with the native operating system and its configuration. The AutoIP and Multicast DNS protocols used by UPnP, which will be discussed later, can actually fit into this void.

3.3 UPNP

UPnP is an extension of the normal PnP, which is known from single stations. The goal of UPnP is to reach a zero configuration network. In this network devices can join and leave without configuration changes and new devices may become aware of other devices or of services in the network. UPnP uses TCP/IP as communication protocol for ensuring a large compatibility.

UPnP defines an open network structure, without any architectural needs (as the Internet). No APIs are defined neither, which allows the programmers to define own APIs.

The Universal Plug and play Forum defines UPnP device and service descriptions (also called Control Protocols or DCP) according to a common device architecture contributed by Microsoft. This forum is also responsible for the standardization. The specification overview is taken from [10].

3.3.1 CONCEPTS

Addressing

As UPnP is based on TCP/IP, DHCP is used to assign an IP address to a device. When no DHCP server is available AutoIP is used. AutoIP chooses intelligently an address of a set of private addresses. If this is used the device can easily change between managed and unmanaged networks. Services in UPnP can use friendly names. To resolve the IP addresses corresponding to these names DNS and Dynamic DNS shall be used.

Discovery

Once an address is assigned, the device can discover control points, or if the device is a control point, it can search for devices of interest in the network. This is done with SSDP (explained later in this document) and an XML discovery message which contains the specific information about the device or one of its services.

Description

After discovery, the control point still knows very little about the device it discovered. Therefore it retrieves a description from the URL given in the XML discovery message. It also gets information of all embedded and included devices of this device.

Control

Once the control point has the description it is ready to control the device. It also has to get a description of all the included services in the device. This description is also in XML and contains variables which express the state. To control a device a control point sends control messages to the control URL for the device. This again is expressed in XML using SOAP. In return to a control message the device sends action specific data or error codes.

Eventing

A PnP description contains state variables which express the state of a service at run time. The service publishes updates, when changes to these variables occur. A control point may be interested in such messages. This is done with event messages expressed in XML formatted with GENA.

Presentation

A device can have an URL for presentation, from which the control point can download a presentation page.

3.3.2 COMPONENTS

Devices

An UPnP device is a container of services and nested devices. Different categories of UPnP devices will be associated with different sets of services and embedded devices. The type of a device and the associated set of services are defined in XML device description documents, with a list of properties that a device must host.

Services

The smallest unit of control in an UPnP network is a service. A service exposes actions and states with state variables. Similar to the device description, this information is part of an XML service description standardized by the UPnP forum. A pointer on these service descriptions is held in the device description. Devices may contain more than one service.

A service consists of a state table, a control server and an event server. The state table models the state through state variables. The control server receives action requests, executes them, updates the state table and returns responses. The event server publishes events to interested subscribers.

Control Points

A control point in an UPnP network is a controller capable of discovering and controlling other devices. It does:

- Retrieve the device description and get a list of associated services
- Retrieve service descriptions
- Invoke actions to control the service
- Subscribe the service's event source

3.3.3 THE UPNP PROTOCOL STACK

Figure 3.2 shows the UPnP protocol stack.

The three upper layers show the parties involved for the provisioning of a service description. To define a service, the UPnP device architecture first defines a schema for creating device and service descriptions for any device and service type. Individual working committees then standardize device and service types and create templates for individual device or service types. Finally, a vendor fills in this template specific to the device or service he offers.

This data is then encapsulated in the UPnP specific protocols defined in the UPnP device architecture document.

The UPnP specific information is then put in all the messages before they are formatted using SSDP, GENA and SOAP.

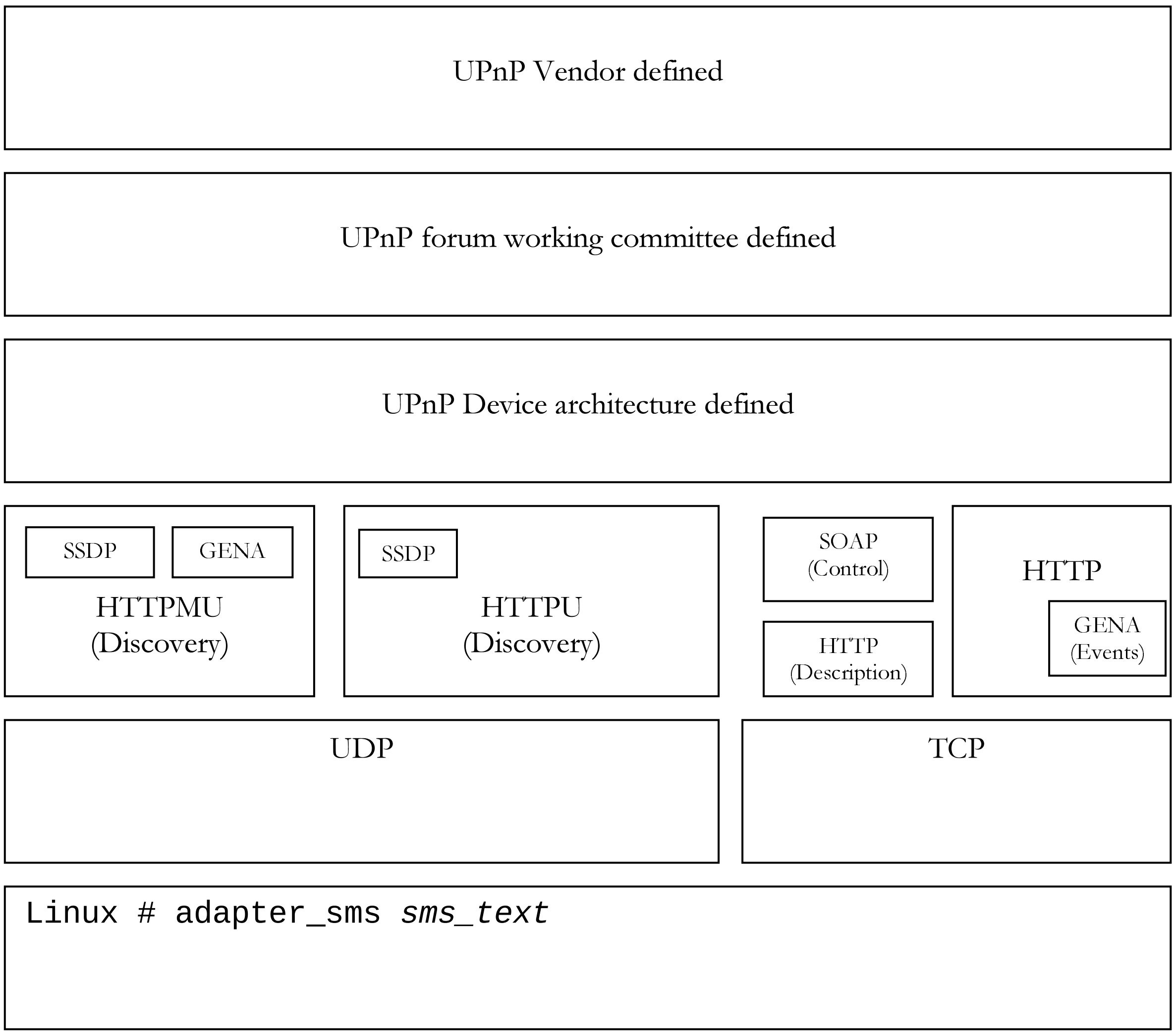


Figure 3.2: The UPnP Protocol Stack

Networking Media for UPnP

As UPnP needs IP to function, the media, which transports IP, is not important for UPnP. Only the bandwidth can limit the UPnP functionalities. UPnP uses standard networking technologies like XML, TCP/IP and HTTP.

TCP / IP

Is the basis of UPnP and ensures that UPnP can be run upon different transport media. UPnP can use many protocols in the TCP/IP stack, like ICMP, UDP and service of it, like DHCP. How this is achieved will be highlighted later.

HTTP, HTTPU, HTTPMU

As HTTP is the base for the Internet success it is a core part of UPnP. All aspects of UPnP are built upon HTTP. HTTPU (and HTTPMU) are variants of the HTTP standard and are used to provide the features of HTTP over UDP/IP instead of TCP/IP. They are used by SSDP, which will be explained next.

SSDP

Simple Service Detection Protocol (SSDP, explained below) is used to discover services in the network. It is used for locating resources and for announcing device availability. Both, control points and devices run SSDP.

GENA

Generic Event Notification Architecture (GENA) [11] was designed to provide the ability to send and receive notifications using HTTP over TCP/IP and multicast UDP. GENA formats are used to send availability notifications used in SSDP and signal channels in service state for UPnP eventing.

SOAP

Simple Object Access Protocol (SOAP) [12] defines the use of XML and HTTP to execute Remote Procedure Calls (RPC). It can also use SSL for securing the communication and it is the standard of Internet RPC. Like RPC UPnP uses SOAP to deliver control messages to devices and return results or errors back to control points.

XML

Extensible Mark-up Language (XML) [13] is the universal format for structured data on the Web. It is the way to put nearly all kind of structured data in a text file. XML is used in UPnP service descriptions, control messages and eventing.

3.4 SSDP

The Simple Service Discovery Protocol (SSDP) provides a mechanism to discover network services by clients with little or no static configuration. Therefore SSDP supports multicast discovery, server based notification and discovery routing.

This chapter about SSDP is a summary of the Internet draft [14].

3.4.1 CONCEPTS

Message Flow on a SSDP Multicast Channel

SSDP clients discover services by using the reserved local administrative scope multicast address 239.255.255.250 over the SSDP port (also referred to as SSDP multicast channel, port). Discovery occurs when a client asks for a service on the SSDP multicast channel/port using HTTP over UDP. A server listens to this channel/port and if one of his services matches the request, it replies to the query through this SSDP multicast channel/port using HTTP over UDP. SSDP services may signal through this channel using HTTP over UDP their presence. For network performance reasons SSDP supports both, service discovery requests and service presence announcements.

The multicast seems to be the right solution to guarantee that a client that is searching for a certain service can discover all the servers in reach providing such a service. Or that the client can inform each other about his presence. On the other hand this mechanism can work without any configuration, management or administration. The administrative scope has been chosen to give the administrator the possibility to choose the machines that are grouped in a unit.

The use of multicast on the other hand can be an issue, as possible deployments of UPnP in large scale networks, like MANs or WANs, can not be achieved, because of the lack of multicast support.

SSDP Discovery Information Caching Model

Services are identified by a unique paring of type URI and name URI (USN). Additionally service discovery results and service presence announcements also provide expiration and location information. Location information tells how to contact a service. The expiration information is like a lease.

3.4.2 COMPONENTS

SSDP Discovery Requests

SSDP discovery requests are extended HTTP SEARCH methods, which serve to discover services in the network. These messages are delivered with HTTP over UDP, but future implementations are expected to support also TCP.

A SSDP discovery request contains a single URI and can specify the service type. Only SSDP services with a matching service type may respond to these requests. Responses to a SSDP discovery request have to be sent to the IP the request came from. Such a response should contain the service location. Responses should also contain information about the expiration of the service. This can be compared to the lease based approach of Jini.

SSDP Presence Announcements

SSDP services may declare their presence on the network by sending a [GENA] NOTIFY method using SSDP presence announcements (ssdp:alive) sent to the SSDP multicast channel/port. When a SSDP presence announcement is received by anybody with a matching USN in the cache, all information in the cache is updated. So SSDP Presence Announcements can be used to update location information and to prevent expirations.

SSDP Presence Announcements must be set to the service's type and must contain the USN of the service. Besides this, they should contain a Location. If no DNS is available at least one IP should be specified.

On the other hand SSDP may notify the clients of the intention to cease by sending a [GENA] NOTIFY using SSDP Presence Announcements (ssdp:byebye) sent to the SSDP channel/port.

SSDP Auto-Shut-Off

There must be an algorithm to prevent that too much network traffic for the network is generated. In a worst-case scenario a huge amount of traffic can be generated by the SSDP. At the moment there is no such algorithm specified in the draft [14]. Nevertheless, this issue has to be handled with care.

SSDP Service Enumeration

All SSDP services must respond to these SEARCH requests over SSDP multicast channel/port as if the request would contain their service type. This can be used to enumerate all the services available on a particular multicast channel/port. This enumerating feature can be useful for monitoring and analysing purposes. And the collected information about all available services can then become available outside the multicast's scope.

3.5 SALUTATION

Salutation is defined by the Salutation consortium and is another approach for service discovery and utilization among the network. With a special attention to the different kinds of devices Salutation proposes a processor, operating system and communication protocol independent solution. The Salutation architecture provides a mechanism for services and devices to describe their capabilities to other services and devices, a mechanism for services and devices to search for other services and devices and a mechanism to request and establish session between them. This chapter is a summary of [15] and [16].

3.5.1 CONCEPTS

Services and Service Descriptions

The concept of a service is broken down into a collection of Functional Units. These Units are representing a special feature, like Fax, Print or sub features of these, like page setup. A service then is made up of a collection of these Functional Units and a service description is a collection of Functional Unit descriptions. These Functional Unit descriptions contain a set of attributes, which are name value pairs. These attributes are queried and matched against queries during service discovery requests. These query and matching requests can be implemented as functions, which define the API for clients.

Registration

Clients can register and un-register themselves with a Salutation Manager (SLM, explained below). This is always done with the local or nearest SLM. When a client registers information the SLM registry is updated and shared with other SLMs. For this interaction the Salutation Manager protocol (see figure 3.3 below) is utilized, which uses Sun's RPC. To discover other SLMs RPC broadcasts can be used or broadcasts dependent on the transport layer and therefore managed by the Transport Manager (TM, see below) internally. This is similar to the Jini lookup service.

Discovery

Clients almost always talk directly to the SLM. Therefore services have to be made discoverable on the different SLMs. To provide this, SLMs discover other SLMs and services registered there. Again, for this the Salutation Manger Protocol is used. For the service discovery the local SLM specifies the service types and service attributes and tries to discover matching entries in other SLMs. This feature is called capability exchange. This capability exchange forms a lookup service distributed over the network.

Eventing

As mentioned the call-backs into the devices from the Salutation can not be used to have the same eventing feature in Salutation like in Jini. To simulate a 'service is now available' event, the client can ask the local Salutation Manager to check periodically for a certain service.

Session Management

A session is established when a client wants to use a service, which it discovered through a service discovery. The communication between client and server can be delegated through a Salutation Manager or not. This is called the Session Management. For this Session Management Salutation defines three modes, in which a session be held:

- Native mode
- Emulated mode
- Salutation mode

In the native mode all the message exchange is done with help of a native protocol and the Session Manager is never involved in the communication.

In the emulated mode, the Salutation Manager Protocol is used for the message exchange. The massages are carried over this protocol, but the content is not inspected.

In the Salutation mode not only the Salutation Manager Protocol is used for the message exchange, even the message format is defined by the Salutation Managers.

3.5.2 COMPONENTS

Salutation Manager

The Salutation Manager (SLM) is the core of the Salutation architecture. It can be compared to the Lookup service in Jini. Services register their capabilities with a SLM and clients can search for other services and devices by sending requests to the SLM. The SLM coordinates requests with other SLMs and establishes a connection channel between the client and the returned service. The behaviour of SLMs can be compared to the one of agents: everything on behalf of the clients. SLMs can even mediate data transfers. This is useful when client and server are using a different communication layer. The framework also provides event messages like data arriving or device unavailable. These events cannot be compared to the Jini event Model.

Salutation Manager Protocol

The Salutation Manager Protocol (SMP) is used by Salutation Managers to communicate. For this protocol Sun's RPC [9] is used. To discover other SLMs an SLM can use RPC broadcasts.

Transport Manager

With help of the Transport Manager (TM) the transport layer independence is achieved. The Transport Manager is dependent on the network transport layer it supports. An SLM can have more than one TM, one for each transport layer, over which the SLM has to communicate. The TM provides transport-independent interfaces to the SLM.

Transport-independent Interfaces

These interfaces ensure the network transparency for the SLM and the clients and servers in top of a SLM. SLMs communicate with TMs through the transport-independent TM interface (SLM-TI) and with clients and servers through a transport-independent API (SLM-API).

Salutation Architecture Overview

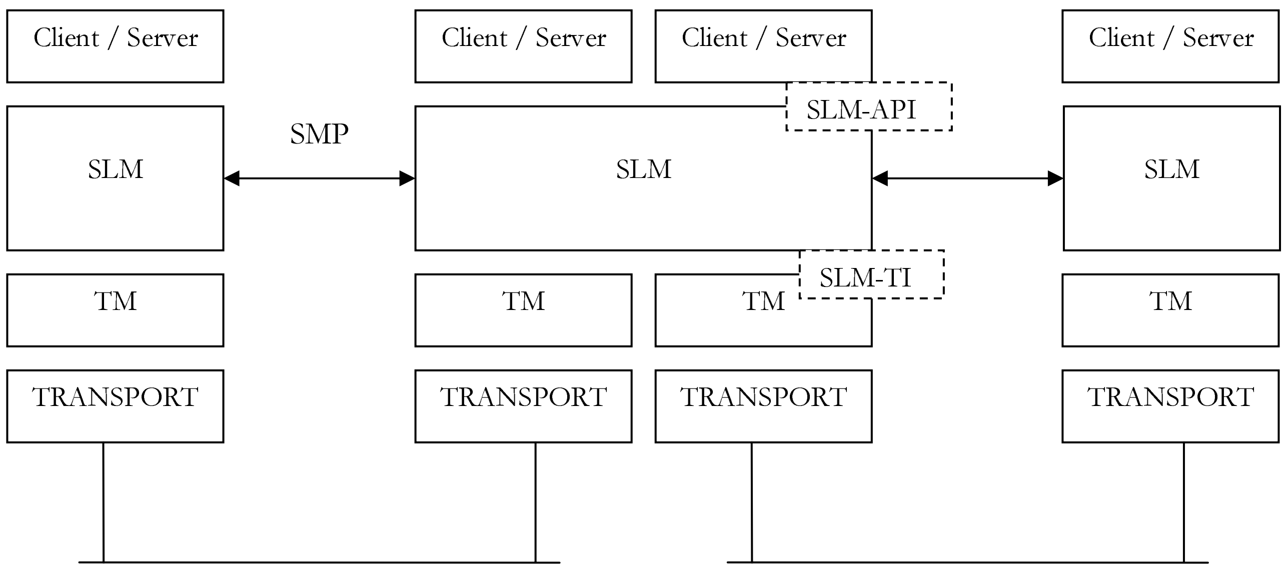


Figure 3.3: The Salutation architecture

3.5.3 SALUTATION LITE – DOWN-SCALED SALUTATION

The Salutation consortium is responsible for the definition of the Salutation Architecture. This is a service discovery and session management protocol. The goal was the definition of an open standard for resource management which is distributed royalty-free.

The consortium detected also the need to support small bandwidths and low battery devices. For this the focus was on wireless networking technologies and on handhelds. So the footprint of the software was the key requirement. Specifications are available under [17].

3.6 BLUETOOTH SERVICE DISCOVERY

About Bluetooth Service Discovery

The Bluetooth SDP is defined by Bluetooth SIG under [19]. Many Bluetooth applications and services are under development at the moment and therefore many new Bluetooth profiles appear. As users can hardly track the whole evolution of new profiles, the Service Discovery Application profile has been defined. It should ease the way services can be discovered and configured with the right parameters. The goal of service discovery in Bluetooth differs from the goals of the above examined service discovery protocols. Auto configuration and building a community wherein services can be announced and discovered by all the members are not essential within Bluetooth service discovery. Much more Bluetooth service discovery is a simple client server scenario in which a client may ask the server about the availability of services on the server and information about how they can be accessed. Configuration is still up to the user or at least to the application intending to use the service. So it can be summarized that Bluetooth SDP provides:

- Search for services by service classes
- Search for services by service attributes
- Browse a device for services

3.6.1 CONCEPTS

Role Playing

For a Bluetooth service discovery procedure at least two devices have to be involved, which play the roles of the service server and client respectively.

The client is the initiator of the service discovery procedure. Thus it must at least implement the client portion of the Bluetooth service discovery profile. This device can discover services and display results of a service discovery process.

The server device is the device which can reply to the service queries and it must at least implement the server portion of the Bluetooth service discovery profile. This device contains a service record database which is consulted by the server portion of the profile to generate responses.

The server and client portions of the service discovery profile can be implemented in the same device and the device can play both of the above roles. But within one service discovery process, the device is always either the service server or client.

Device detection, service discovery and connections

To initiate a service discovery procedure a device first has to detect other Bluetooth equipped devices within its range. Then a connection to the detected devices has to be established with Bluetooth conform security aspects (e.g. by providing a PIN). After the connection establishment the client can do service discovery queries over the created link.

3.6.2 COMPONENTS

Service Discovery Server

The service discovery server implements the function primitives to interact with the server's database and the protocol data units (PDU) to interact with the service discovery client over a Bluetooth link. The function primitives are necessary to register a service description and to do queries in the server's database to answer to discovery requests and the PDUs are necessary to communicate via the Bluetooth link with a Bluetooth service discovery client.

Service Discovery Client

The client side of the Bluetooth service discovery profile implements the function primitives for the application to query a remote device for available services and service attributes. On the other hand it provides the PDUs to interact with the service discovery server over a Bluetooth link.

Service Discovery Application

The application uses the interface, the client part of the Bluetooth service discovery profile implementation provides, to do queries to a remote device. The application can also implement the Inquiry and connection establishment to ease a service discovery process. The application is used to:

- Search for services by service classes
- Search for services by service attributes
- Browse for services

on the remote device and to display the results via a user interface or to establish a connection to the service. With help of the application some of the principles of Jini, UPnP and Salutation can be simulated (like Auto configuration, service enumeration and service discovery). But this approach is still a very basic approach and cannot fulfill the same tasks as the former mentioned service discovery protocols.

Service Descriptions

Service descriptions are done by filling in description templates. For known services there exist already predefined templates. They are built of attributes, which are name value pairs or a collection of other attributes. The predefined service templates can be seen in Bluetooth Assigned Numbers – Service Discovery Protocol [20].

3.6.3 THE PROTOCOL STACK

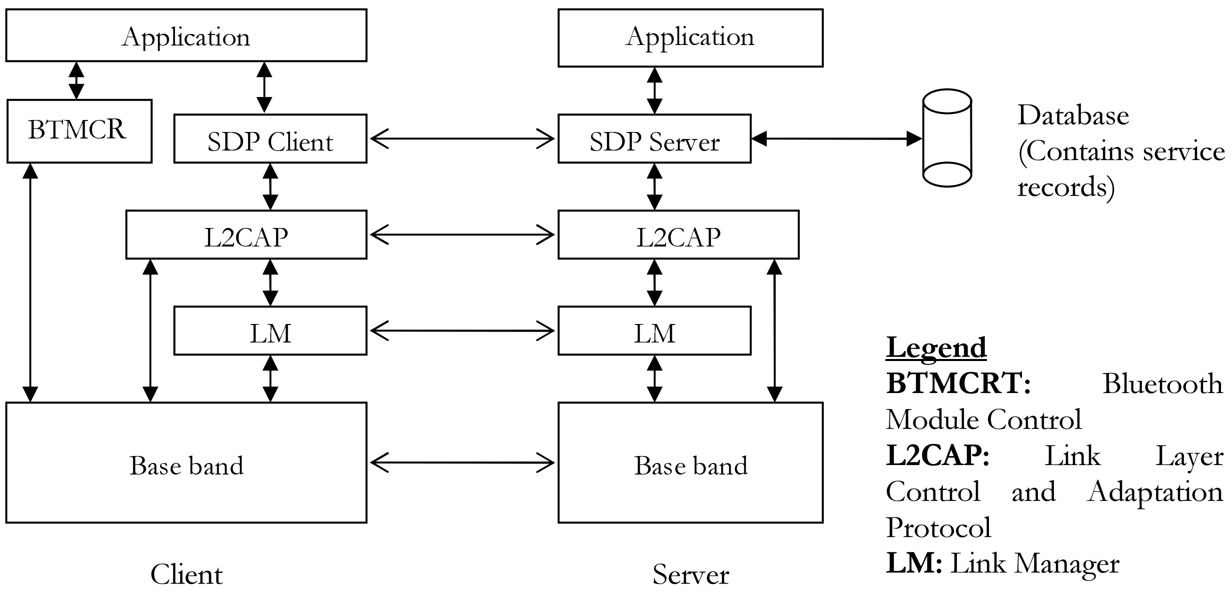

Application
BTMCR
SDP Client
L2CAP
LM
Base band
Client
Application
SDP Server
L2CAP
LM
Base band
Server
Database (Contains service records)
Legend
BTMCRT: Bluetooth Module Control
L2CAP: Link Layer Control and Adaptation Protocol
LM: Link Manager

Figure 3.4: The Bluetooth SDP Stack

Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer

As explained above, the Salutation architecture is a very flexible approach for service discovery. It is independent of the transport layer. This chapter describes how the Salutation architecture and the Bluetooth Service Discovery Protocol can be merged. This would have the advantage that also Bluetooth devices could fit perfectly in a Salutation environment. The full suggestion can be accessed under [21].

To map the two service discovery protocols, there can be two approaches. The first approach will assume that the Salutation APIs are implemented on top of the Bluetooth Service Discovery Protocol and Bluetooth SDP attributes pass through the Salutation API.

The second approach will assume that a Salutation Manager can directly communicate over Bluetooth links, with an implementation of a Bluetooth specific Transport Manager. In this approach the whole Bluetooth SDP layer would be replaced by the Salutation Manager, which maps its functionality to the Bluetooth Service Discovery Protocol.

3.7 SERVICE LOCATION PROTOCOL

The Service Location Protocol (SLP) is being developed by the IEFT SrvLoc working group [22]. With help of SLP computers use little or no static configuration of network services for network based applications. SLP is a decentralized, lightweight, scalable and extensible protocol for service discovery and service selection. Traditionally users had to find services by knowing the name of a network host or an IP address providing the desired service. SLP eliminates the need for a user to know about the name or the address of the host providing the service. Rather the user supplies the desired type of service and a set of attributes describing the service. Based on this information SLP resolves the name of a host providing the needed services.

SLP is designed to work in networks under a corporative administrative control. This allows providing security, multicast routing and organization of clients and services, which would not be possible in the whole Internet. Therefore SLP has been designed to serve enterprise networks with shared services and it may not necessarily scale for WANs. The SLP definitions are accessible in [23].

3.7.1 CONCEPTS

In the definition of the SLP user applications are modelled as User Agents (UA) and services are provided by Service Agents (SA). For reasons of scalability Directory Agents (DA) are used. With these DAs services can be registered and services can be discovered.

When a user application needs a service, the UA issues a service request (ServReq) on behalf of the client application, specifying the characteristics of the needed service. As answer the UA will receive a service reply (SrvRply) containing the location of all the services satisfying the request.

Service Requests

The simplest case is when an UA issues service requests directly to a SA. In this particular case the request is multicast over the network. SAs receiving a request for a service send replies via unicast to the UA with the location of the service. In larger network this process does not scale and therefore Directory Agents are used. DAs have the functionality of a cache. SAs register services with a DA and UAs contact DAs for service requests. To do so UAs can unicast service requests directly to a DA instead of multicasting them over the network. The DA will answer to the request with a unicast service reply containing the location of every service matching the request. If no DA is known, the UA will try to discover one.

Service Registration

SAs can send service register (SrvReg) messages for the services they advertise to a DA and receive acknowledgements (SrvAck) when the registration of a service with a DA was successful. Both, the Registration Request and the Registration Acknowledgement are unicast. The service registrations have to be refreshed periodically or the service registrations on the DA will expire. If a SA does not know any DA it will try to discover one.

Directory Agent Discovery

UAs and SAs can discover DAs in two ways. First they can multicast a service request over the network and second by receiving presence announcements the DAs send out periodically and the UAs and SAs are listening for. In both cases the UA respectively the SA will receive a DA advertisement (DAAdvert), which in the first case is unicast to the requesting agent and in the second case multicast over the network.

Scopes

Services can be grouped together by using scopes. These scopes are strings identifying the services. With help of scopes locations can be identified, administrative grouping can be done or proximity in the network topology can be modelled. SAs and DAs are always assigned a scope. This means that they will only answer to service requests in their particular scope. They can also be assigned more than one scope. UAs can contain a scope, which means that UA will only be able to discover services within that scope. With help of such a scope the administrator can make a basic assignment of services to users.

Security Aspects

SAs and UAs can verify digital signatures provided with DAAdverts. UAs and DAs can verify service information registered by SAs. The security parameters to verify these signatures are specified in the SLP Security Parameter Index (SLP SPI) and passed within Authentication Blocks, which extend the normal SLP protocol. The interested reader may find the definition of authentication block in [24].

Service URLs

A service URL indicates the location of a service This URL is conforming to the URI standard, but SLP always uses address specification. Every network service can be defined with a service URL.

3.7.2 COMPONENTS

The User Agent

As mentioned the User Agent takes the part of the client. In order to have a working SLP implementation an UA interacts either directly with a Service Agent or when a Directory Agent is present with the Directory Agent. For this interaction an UA must be able to send service requests and to interpret DA Advertisements, SA Advertisements and service replies. Service requests are issued following these rules:

1. UA issues a request to a DA which it has been configured with via DHCP

2. UA issues a request to a DA which it has been statically configured with

3. UA uses multicast service requests to discover DAs

4. UA uses multicast service requests to discover SAs

An UA can be configured with a scope using DHCP or static configuration. No configuration means the scope "Default" and the UA will detect its scope list automatically. The UA must issue service requests with at least one of the configured scopes.

The Service Agent

The Service Agent takes the role of the service provider. In an SLP implementation a SA must be able to answer to service requests either with service replies or with service advertisements. An SA has to register its services with a DA as soon as DAs are detected, when they are serving the same scope as the SA is configured with. Like UAs SAs can be configured with a scope following the same rules. In addition to a UA an SA must listen to unsolicited DA Advertisements to become aware of available DAs.

The Directory Agent

In an SLP implementation the Discovery Agent takes the role of some kind of a cache. The DA can be thought as a relay component between the UA and SA. An SA registers all its services, which match the scope(s) the DA serve(s), with a nearby DA. The SA can be configured with DA addresses or the SA can try to discover DAs. The UA in turn searches on a nearby DA for the appropriate services. Also here, UAs can be configured with DA addresses or it can detect DAs. The Directory Agent must therefore be able to accept unicast requests and multicast Directory

Agent Discovery service requests and it must be able to respond to UDP and TCP requests as well as to multicast DA discovery service requests. In addition the DA can send out unsolicited DA Advertisements to signal his presence to SAs and DAs in the network. The DA is not necessary for a basic SLP implementation, but an implementation without DAs will not be scalable. If possible, DAs should be used.

3.7.3 RESOURCE DISCOVERY PROTOCOL

The Resource Discovery Protocol (RDP) originates from an approach to solve the problems that exist in a Mobile IP based infrastructure. In such a structure the client does use the resources provided in the home network in an application-transparent way. The approach of the RDP was to use local resources. To do so, a protocol was defined, which can be seen as the basis of the SLP. In RDP the three components UA, DA and SA are defined. The SA is the resource provider, the DA a kind of repository and the UA is the resource client. The RDP is rather a proof of concept as a protocol and therefore there are some basic problems. First of all the RDP does only define the use of one DA which's address is provided by DHCP. The SA is registering resources with this unique DA and the UA is requesting resources from this unique DA. This is applicable for small networks with just a handful of shared resources, but it does not scale for larger networks. As a second problem the security has to be mentioned. The address of the DA is stored in a plain file, to make it easily accessible for the applications requesting resources. The address is also stored on a bad guy's snooping machine, which then can identify resource request, which can help to construct Denial of Service (DoS) attacks. The third weakness of the approach is resided in the fact that exclusively UDP packets are used. As UDP packets are fixed sized, and resource request can result in a list of available resources, the size of UDP packets can be to low to hold the whole response.

As mentioned the RDP serves as a basis for the SLP. The work of the RDP has been taken over from the IETF SrvLoc group, developing the SLP. The SLP solves the weaknesses of the RDP. For the scalability issues in SLP the use of no or more than one DA is allowed. Therefore the SLP does scale for very small networks as well as for large ones. This made it unavoidable to solve the UDP problem as well, as SA or DA search requests could cause responses which are too big for a UDP packet. So the use of TCP is also defined in SLP. As an addition the SLP does define multicast, which has to be used to detect DAs respectively SAs which can not be longer configured only by using DHCP. With help of multicast scopes can be used which offers a weak security. Again, the use of multicast may prevent SLP deployment in large scale networks like MANs and WANs. For a detailed comparison between RDP and SLP the document [25] can be consulted.

3.8 CONCLUSIONS

Service discovery protocols are defined to facilitate dynamic operation of network devices in changing network scenarios without or with little user interaction. To achieve this dynamism, devices in service discovery protocol environments must be detectable in the network, must be able to discover services in the network and must be able to access these services. All the presented service discovery protocols enable these features.

Jini has a forced dependence on the Java Virtual Machine. Therefore all devices, taking part in the network, have to support Java. With the principle of the Jini-proxy, this need can be avoided, especially for devices with limited resources, like cell phones. This Jini-proxy principle is one of the strongest advantages of the Jini specification, not found in the other systems. But the proxy has an impact on the service or device manufacturer as they have to use all the same standard interface for the same type of devices or services to enable the proxy possibilities. Unfortunately Jini does not exist in an open source implementation, which can make it hard to adopt Jini to be integrated with CAHN.

UPnP has its dependence on the existing IP and Web technology. Therefore integration in today's networks should be rather easy. UPnP focuses more on strong description of device capabilities, on distributed control of the devices and on eventing mechanisms. UPnP can be seen as an addition to the known PnP principle. For that reason UPnP fits better in home networks than in large-scale networks.

SSDP is a part of the UPnP definition and is used in combination with this system. It defines solutions for some of the requirements of the UPnP system and addresses the same overall purpose.

The Salutation Service Discovery Protocol has its focus on the service discovery problem and on the session management. After all, Salutation's transport layer independence make Salutation the most flexible and scalable solution for service discovery. The downscaled version of Salutation, Salutation-Lite provides the same approach for low-resource systems. The focus of Salutation-Lite is on a small footprint of the implementation and on the additional Operational Environment and Display Functional Units to give the service provider the possibility to adapt the services according to the client's device capabilities.

The Bluetooth Service Discovery Protocol is defined as a communication channel between a client and a server device. The goal is to give a client the possibility to browse a detected device for available services. Each device is able to be client or server, depending on the situation. With help of an integration of the Salutation system into the Bluetooth Service Discovery Protocol the system can be expanded and offer additional features. The Bluetooth SDP itself is a basic approach to service discovery, but it has reached a productive state and can be an enabler for dynamic networking.

Similar to Salutation the Service Location Protocol (SLP) focuses pure service detection. Unlike Salutation SLP was defined to work in corporate networks and has no focus on scalability for larger networks, even if better scalability could be achieved with some clues. SLP can work as a pure client server approach and with help of Directory Agents (DA) as well in distributed systems. DAs are needed by the system to scale. SLP offers a very clean and simple definition for service discovery. OpenSLP is an open-source implementation of SLP, which already made his way into several recent GNU/Linux distributions.

Compared with each other the Jini approach has his strength in the support for proxies, so that clients of services do not have to install supplementary drivers to access a service. Another advantage is the JAVA basis which offers JINI a high acceptance. UPnP does address the topic of service discovery from the point of view of the network components, instead of the service itself. Therefore the UPnP approach does not scale for big networks. Salutation offers this scalability, by providing mechanisms, how different SLMs can talk together. So the whole system is distributed over several SLMs, which makes the system very flexible. Moreover Salutation also focuses on the applicability on devices with limited resources. The flexible design of Salutation also allows integrating Bluetooth SDP into the system. The Bluetooth SDP itself focuses on the use of the protocol to discover services on other Bluetooth equipped devices within the Bluetooth radio range, which is very limited. Therefore it makes no sense to define a separate catalogue server, where devices can register their services. So the catalogue is kept on the Bluetooth device, which supports the services. Last but not least, the SLP offers a clear and straight forward definition of service discovery. SLP does only address service discovery in LANs and corporate environments. The use of Directory Agents is not mandatory so that SLP can be used in a comparable manner to Bluetooth SDP, where the services are only advertised by the device providing the service.

Table 3.1 illustrates the comparison of the different service discovery protocols.

	Main Purpose	Dependencies	Security	Particularities
Jini	Providing a distributed system, which is easy to administer and where federations of users and their resources can be built.	> Java > IP networking	> Specialized Jini-focused solutions > Use of principal and access list to control permissions	Defines the use of Proxies and RMI to provide the communication between the different entities. Therefore client software is not necessary to access a service.
UPnP	Extend the known PnP mechanisms to the network, so that no configuration is needed when new network components are added or existing network components are removed.	> Multicast support > IP-networking > web standards (i.e HTTP, XML …) > SSDP	> Only scope mechanisms of multicast can be used for access restrictions	Tries to rely on web based standards to achieve an easy integration into existing networks. Does not focus large-scale networks. Seperate Service Discovery Protocol: SSDP
Salutation	Proposes a processor, operating system and communication protocol independent solution with a special attention to the different kinds of devices.	No special preconditions		Special down-scaled solution exists with small footprint. Bluetooth SDP can be integrated in Salutation. SLM interoperation grants scalability. Very flexible, but very complicated specifications
Bluetooth SDP	Providing a mechanism to find Bluetooth services in radio range and get information how to get access to these services, with optimization for the usage by devices with limited resources.	> Bluetooth	> Bluetooth Security	The Bluetooth SDP has no specialized service catalogue. The services are registered on the device providing the services.
SLP	Providing an environment, where devices can use services with little or no static information. With a special focus on corporate LANs.	> Mulitcast support > IP networking	> Only scope mechanisms of multicast can be used for access restrictions	Very clear and compact definition of Service Discovery Protocol. Service Catalogues (Directory Agents) are optional. Not scalable, because of multicast usage.

Table 3.1: Comparison of service discovery protocols

Finally it has to be drawn the conclusion that the presented service discovery protocols share two main issues. The first issue is the lack of security. In the specifications of the service discovery protocols is mentioned little or nothing about security. The suggested security approaches all prerequisite prior registrations of the devices to a certain service discovery protocol system, so that they can be authenticated later. This is not the desired solution, as service discovery protocols should provide environments, where devices can participate with little or no static configuration. The use of certificates on the different components can be a possible solution, but the therefore necessary checks of the certificate revocation lists require Internet access. This can not be guaranteed in all environments, i.e. Bluetooth environments.

The second issue that has to be mentioned here is the strong dependence of some service discovery protocols on multicast support. Even if multicast support can be found in corporate networks, large-scale network often do not provide support for it. This often reduces the scalability of a service discovery protocol and is a serious issue for a possible deployment.

4 CELLULAR ASSISTED HETEROGENEOUS NETWORKING (CAHN)

4.1 THE BOOTSTRAPPING PROBLEMS

Link Establishment

In heterogeneous environments the setup of a connection between two nodes can become quite complicated. As many different access technologies may be involved, the setup of the network quickly becomes confusing. This makes it hard, for the ordinary users to communicate with each other. Furthermore, if the connection has to be secured between the nodes, additional parameters have to be exchanged, and the configuration has to be adapted accordingly.

Authentication

Once two users have managed to setup a connection between their devices, they have to identify and authenticate their devices in order to establish a secured connection. This is crucial, as the two users have to be sure, that they really establish a secured connection with the indented nodes. This makes it necessary for the users to prove, that a certain device really belongs to the right user. The easiest way to do so is by using a shared secret, which have to be known by the users to prove their authorization. The agreement on such a shared secret can be done verbally, but it is important, that this agreement is kept secret from the public, and can not be done over an unsecured link. It is also obvious, that the setup for later authentication processes has to be done prior to the authentication process itself. Therefore a proper authentication setup can be seen as the enabler for a secured link. Again, the existence of different authentication mechanisms makes it almost impossible, to find a common authentication scheme, which is applicable for all the users in a heterogeneous network.

Secured Link Establishment

After the authentication, the link between the users has to be secured. For that purpose the users have to exchange several security parameters, which do at least consist of the encryption algorithm and the encryption key. To setup more sophisticated encryption mechanisms, further parameters have to be negotiated. In any way, the users have to perform a handshake in order to establish a secured connection and this handshake must also happen in a secure way.

4.2 AUTHENTICATION AND SECURED LINK ESTABLISHEMENT BY CELLULAR OPERATORS

In the CAHN description [25] it is suggested to use a cellular network to assist the users in a heterogeneous network. The cellular network thus serves as a channel which can be used for the signaling. The exchange of the needed parameters for a connection establishment and further the negotiation of the needed parameters to create a secured connection can be handled over such a channel. The typically high coverage of a cellular network makes it very valuable for the use as a signaling channel.

Besides the provisioning of a signaling channel, the cellular network offers an existing trust relation between the users of a cellular network and its operator, which can be reused to provide authentication.

Therefore the operator will serve as a signaling and configuration provider for heterogeneous networks. It can locate the peers, authenticate them and offer a configuration for the secured connection establishment.

In order to integrate the cellular operator to assisted users in a heterogeneous network, interaction with the cellular network is necessary. Further the authentication of users in cellular networks is most likely based on SIM mechanisms, which require the interaction with the operator. But not every device in the heterogeneous network has an interface to the cellular network. Devices which have an interface and thus access to the cellular network are referred to as Cellular Aware Nodes (CAN) and devices which have no access are referred to as Non Cellular Aware Nodes (NCAN).

In order to integrate and enable the participation of NCANs in the system, CAHN proposes a relaying of CAHN messages over a CAN. Based on the assumption, that users possess at least one cellular capable device the formation of a Personal Area Network (PAN) for all the devices of a user is suggested. In this network, every NCAN has a connection to the CAN, which can relay the CAHN related messages over the cellular network. Figure 4.1 illustrates such a PAN.

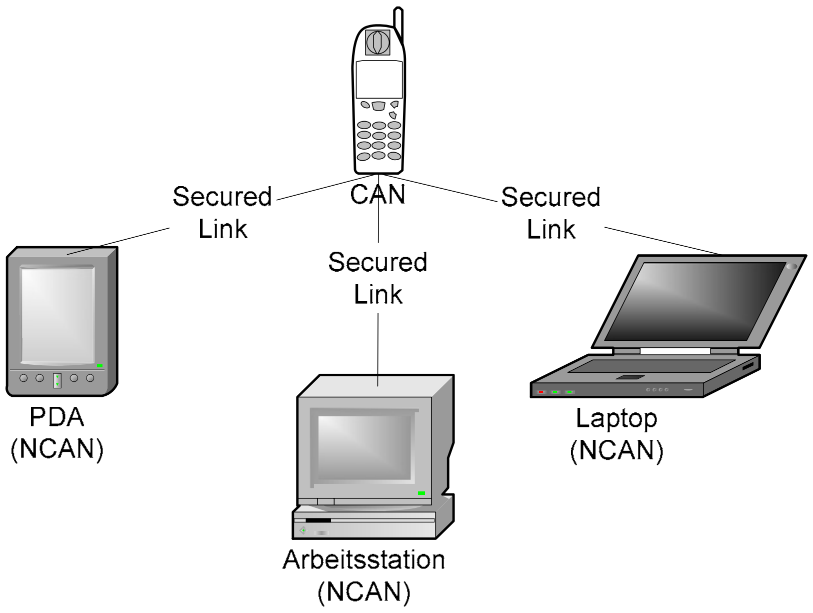


Figure 4.1: Personal Area Network of CAN and NCANs

4.3 THE CAHN TARGET SCENARIO

In its target scenario CAHN can enable networking among nodes with heterogeneous access technologies and furthermore it can assist in the establishment of secured connections among these participating nodes. The presence of different technologies and the increase of complexity in heterogeneous environments make the connection establishment among their devices difficult for the users. CAHN can help in these cases, when it can collect information on the present technologies. With help of this information a network topology can be computed, where all the nodes are connected. How this calculation is done, is not defined, yet. Several approaches are imaginable: from decentralized approaches using agent technologies up to the fully centralized solution, where the cellular operator calculates the network design. Figure 4.2 shows an example of a target scenario.

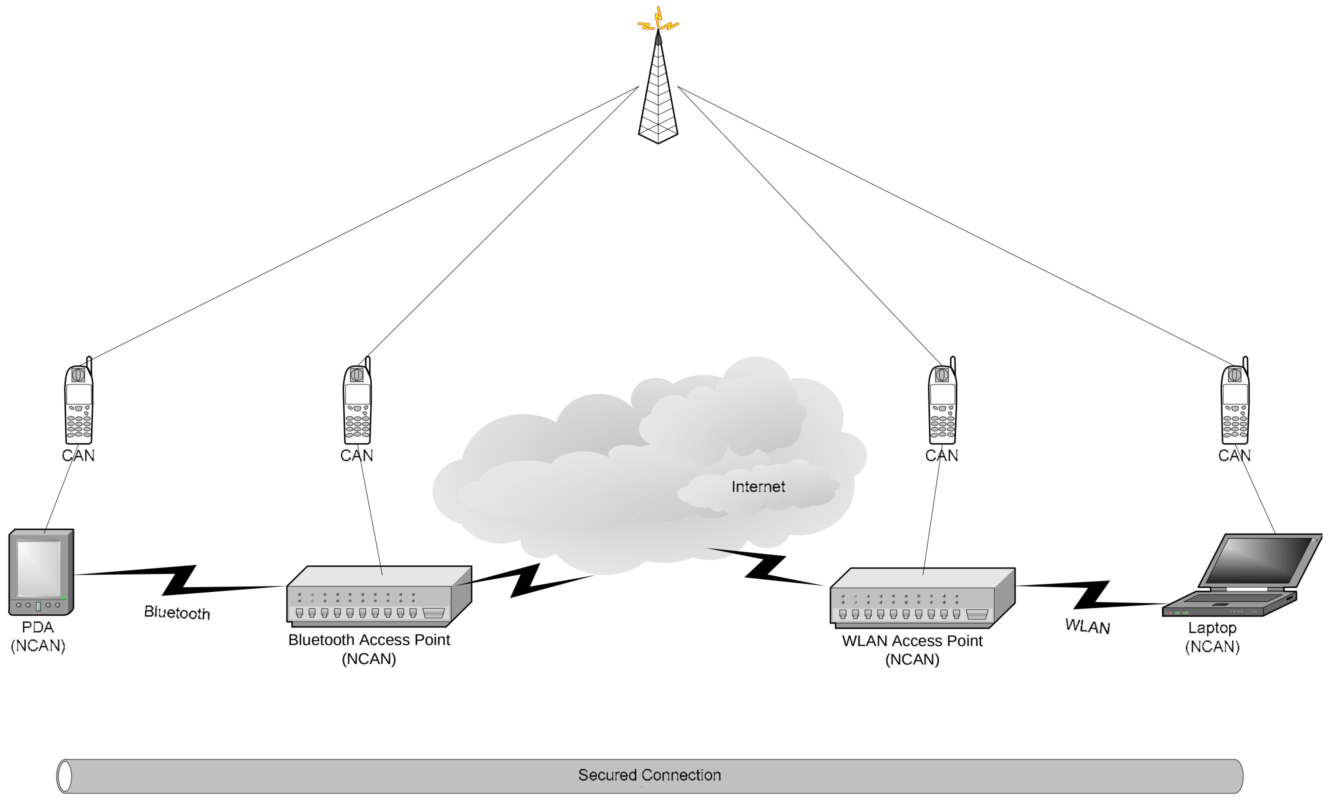


Figure 4.2: A CAHN target Scenario

4.4 CONCLUSIONS

CAHN is a very promising approach. In its basic functionality it can provide an authentication feature, which is almost pervasive.

Further, the provided signaling can ease the configuration of the different networking technologies present in a heterogeneous environment and can help to setup a secured connection between users of the heterogeneous network.

To enable CAHN, devices must have the possibility to access the cellular network. As not every device in the network will have an interface to the cellular network, CAHN proposes to relay the CAHN messages over a CAN and to form a PAN of all the devices a user possesses.

The requirement that per person at least one cellular subscription must be present is not too heavy, since almost every road warrior possesses such a mobile contract. Even if this would not be the case, the price for a subscription and the costs for the exchanged information are bearable.

The problem of identifying devices in heterogeneous environments is not present in CAHN, as all users are identified with their mobile phone number (MSISDN). Upon connection establishment the nodes exchange their configuration parameters over the cellular network and thus also the present connection identifiers of the data link (i.e. the IP address of a WLAN connection). For that purpose it becomes possible to authenticate the participant and with help of the signaling also his device.

5 DESIGN

As stated in the second chapter, the Bluetooth security can be very limited. In the case where no former trust relation exists between the users and no common PIN has been defined the Bluetooth security can not be applied and the connection has to remain unsecured. This is a serious issue in public access scenarios, where security is a must.

To enable Bluetooth in public scenarios, enhanced authentication features are necessary, which are also applicable without having agreed on the shared secret (PIN) in prior.

CAHN can now offer this enhanced authentication, which is based on the trust relation of the user with the cellular operator. With help of this authentication and the secured channel, that CAHN offers, the PIN can be negotiated dynamically, and the Bluetooth link can be secured, using the built-in Bluetooth security mechanisms.

The second part of this diploma work aims at integrating CAHN in the Bluetooth SDP. With help of this integration, the applicability of the Bluetooth security can be improved, and on the other side, the Bluetooth SDP can be used to learn about the CAHN capabilities a certain device can offer.

To show the benefits of this integration, a demonstrator will be implemented. This demonstrator provides basic implementations of CAHN components and the CAHN protocol, and the integration of CAHN as a service into the Bluetooth SDP.

As usability is one of the key motivations for this diploma work, an easy-to-use Graphical User Interface (GUI), which will act on the behalf of the user, will round up the implementation.

Last but not least, the demonstrator is used for measurements which will provide practical knowledge about CAHN. These empirical parameters are important for the further development of CAHN.

5.1 REQUIREMENT SPECIFICATIONS

The main purpose of the implementation of the integration of CAHN in the Bluetooth SDP is to enhance the Bluetooth security and to provide a solution for the bootstrapping problem of Bluetooth in public environments. Further, the Bluetooth SDP can enhance the applicability of CAHN and can be reused to indicate and learn about the supported CAHN features of a certain device. To meet these goals, the following requirements can be defined:

- CAHN must be integrated as a service in the Bluetooth SDP. This indicates the capability of a certain device to support CAHN and to indicate the CAHN configuration parameters.

- CAHN must be implemented as a service attribute to point out that a certain service is enabled for CAHN. This would bring in the flexibility to use Bluetooth services normally or with CAHN enhancements.

- The CAHN core must provide a server implementation to handle connection requests and generate responses.

- The CAHN core must provide a client implementation to generate connection requests and to handle responses.

- The CAHN core implementation must be able to configure the Bluetooth interface.

- Devices must be able to act as a server or a client, according to the situation and therefore the CAHN core must provide the server and the client implementation.

- The CAHN core implementation must provide the mechanisms needed to send/receive messages over/from the used cellular network.

- A CAHN protocol has to be defined, for the CAHN core server and CAHN core client interaction.

- Finally an easy-to-use GUI must be provided.

5.2 DEFINITION OF THE CAHN BLUETOOTH SERVICE AND THE CAHN BLUETOOTH SERVICE ATTRIBUTE

5.2.1 SERVICES IN BLUETOOTH SDP

Bluetooth services are entirely described by sets of attributes. These sets are stored in service records which are maintained by the Bluetooth SDP server. Attributes are name/value pairs. The name of an attribute is represented as a 16 bit unsigned integer, which makes the attribute distinguishable within a service record. Values of attributes can be of different basic types like integers, strings, Booleans or a sequence (list) of those. Each service is defined by at least two mandatory attributes, which are the Service Record Handle (AttributeID: 0x0) and the Service Class ID List (AttributeID: 0x1). The Service Record Handle is a 32 bit unsigned integer and it uniquely identifies each Service Record on a Bluetooth SDP server. The Service Record Handle with the value 0x0 is the Service Record Handle of the Bluetooth SDP service, which must be present on each Bluetooth SDP server and contains further details about the Bluetooth SDP service. Service Records and thus services are organized hierarchically. This means, that a service can be a sub service of a given service and add more details to his super service. For example a printer service may be present on a Bluetooth SDP server with a sub service duplex printer, and another sub service colour printer. A duplex colour printer service would then be a sub service of duplex printer and a sub service of colour printer. For that reason the Service Class ID list contains a sequence of Service Classes, a given service belongs to. Furthermore the Service Class of a service defines the attributes contained in a Service Record for a specified service. Each service must belong to at least one Service Class. Service Classes are referred to with the help of Universally Unique Identifiers (UUID). The interested reader may find UUIDs for Service Classes in the "Bluetooth Assigned Numbers" [20]. Besides the mandatory Service Record Handle attribute and the Service Class ID list attribute typically three other attributes are defined: the Service UUID (AttributeID:0x3), which's value is a 128 bit value that uniquely identifies a certain service among all namespaces, comparable with Service Class UUIDs; the Protocol Descriptor List (AttributeID: 0x4), which contains a sequence of protocol IDs that are used for this service ; and the Human Readable String (AttributeID: 0x100), which's value is a string describing the service in a human readable way.

5.2.2 DEFINITION OF THE CAHN SERVICE

As explained in the previous chapter each service must be defined by at least two attributes, which are the Service Record Handle and the Service Class ID List. The Service Record Handle is generated according to the specific implementation of the Bluetooth SDP profile. As the CAHN service does not fit into an existing Service Class ID (remember: the Service Class ID defines the attributes present for a specific Service Record) a new Service Class with the ID 0x1220, which is not yet taken for another Service Class according to "Bluetooth Assigned Numbers", has been chosen. This Service Class does now define the CAHN service.

- CAHN Service Class ID: 0x1220

In addition to the mandatory attributes certain existing attributes have been defined within the CAHN Service Class: The Protocol Descriptor List (AttributeID: 0x4), which is empty, as no Bluetooth protocols are used in the CAHN service; the Human Readable String (AttributeID: 0x100) with the string value "CAHN enabled device"; and the Service UUID (AttributeID: 0x3) with a value of 0x1220. Again this UUID was not assigned to another service according to "Bluetooth Assigned Numbers".

- CAHN Service UUID: 0x1220

- CAHN Protocol Descriptor List: 'empty'

- CAHN Human Readable String: "CAHN enabled device"

Last but not least the purpose of the CAHN service is not only to indicate the CAHN capability of a given device, but also to enable a connection using CAHN. Therefore at least another important attribute has to be added, to indicate the connection parameters, which in this case only consist of the MSISDN of the given device. The new defined attribute has been assigned a free ID (according to "Bluetooth Assigned Numbers"): 0x400 and its value is of the type String:

- CAHN MSISDN: AttributeID:0x400, value type: String, value: current MSISDN

With these definitions the CAHN service can be integrated in the implementation of the Bluetooth SDP. How this is done in detail will be explained after the definition of the CAHN service attribute.

5.2.3 DEFINITION OF THE CAHN SERVICE ATTRIBUTE

The CAHN service attribute is intended to be used to indicate the ability of CAHN of a given service. This adds more flexibility to the implementation. Therefore, a new attribute with the available ID 0x401 has been added, with a possible value of the type string. The exact value of the attribute does not matter, as the pure presence of the attributes already indicates that the service is CAHN enabled. In this implementation the value contains the Service Record Handle of the CAHN enabled Service to facilitate Bluetooth SDP operations regarding that service. In practice it would be better to assign just a Boolean value to this attribute, in order to reduce the submitted data.

- CAHN service attribute: AttributeID: 0x401, type: String, value: does not matter.

5.2.4 DEFINITION OF A CAHN PROFILE

In order to have the CAHN service and the CAHN service attribute integrated in future Bluetooth implementations the definition and standardization of a CAHN profile through the Bluetooth SIG would be necessary. As the implementation and the definition were not in a final state at the time of writing, this profile cannot be realized yet, but can be a future topic. Nevertheless, a Profile ID was chosen and will be integrated in the used Bluetooth SDP implementation.

- CAHN Profile ID: 0x1220

5.3 CAHN PROTOCOL DEFINITION

The basic CAHN protocol must provide at least three kinds of messages: a connection request message, a connection response message and an error message.

5.3.1 CAHN BLUETOOTH SERVICE REQUEST

This first message is needed to invoke a connection establishment. With help of this message the client can request a connection with the server. Therefore the message is also used on server side to identify the client and must thus contain the identifiers of the client. As the application is intended to handle security per service, the client must also include the ID of the service to which he wants to connect. Figure 5.1 shows the format of a Connection Request message.

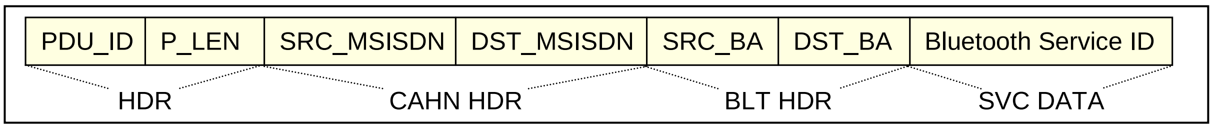


Figure 5.1: CAHN Bluetooth Service Request

For a CAHN protocol packet three different kinds of header can be distinguished: the global header (HDR) is used for packet identification and transmission; the CAHN header (CAHN HDR) contains the information needed to identify the CAHN communication partners; the third header (BLT HDR) is the header that is needed in respect of the service. As this application is based on Bluetooth, this header contains the Bluetooth related parts. What follows the headers is the service related data (SVC_DATA). The following list contains a brief explanation of the different fields of the CAHN Bluetooth Service Request message depicted in Figure 5.1:

- The PDU_ID uniquely identifies the CAHN protocol message among all the CAHN messages.
- The P_LEN indicate the length of the packet and is needed for packet reception, to allocate an appropriate amount of memory for the packet.
- The SRC_MSISDN holds the MSISDN of the packet originator and
- the DST_MSISDN the MSISDN of the receiver.
- The SRC_BA contains the Bluetooth address of the sender and

- the DST_BA the Bluetooth address of the receiver (this address must have been detected before).

- The SVC_DATA comprises all the information that is necessary for a certain service (in this case this field shows the Service ID, the user intends to connect to).

5.3.2 CAHN BLUETOOTH SERVICE RESPONSE

In order for a server to allow a connection and to give a positive answer upon to connection request, a response message is needed. This message must contain the identifiers of the server, so that the server can be authenticated on client side. In addition to that the message must include the PIN that has been generated on server side for the connection request. Figure 5.2 shows such a response message.

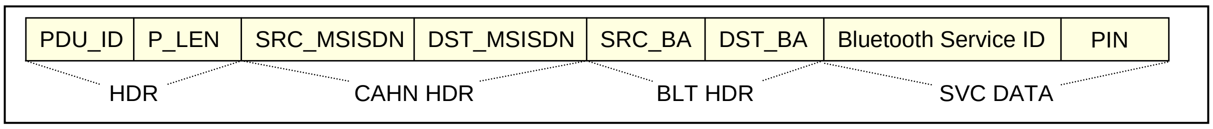


Figure 5.2 CAHN Bluetooth Service Response

The response message shown in the illustration does not differ much from the request message. The fields are again the same as defined for the request. The only addition in the response message is the PIN field in the service data section.

- The PIN field contains the PIN that has been generated for a Bluetooth Service Request on server side.

5.3.3 CAHN ERROR RESPONSE

Not every connection request will lead to an accepted connection. In case of failure, the CAHN Bluetooth Service Response cannot be used to indicate the reason that caused the error. For that purpose error messages have to be defined. Figure 5.3 contains a scheme of a CAHN Error Response.

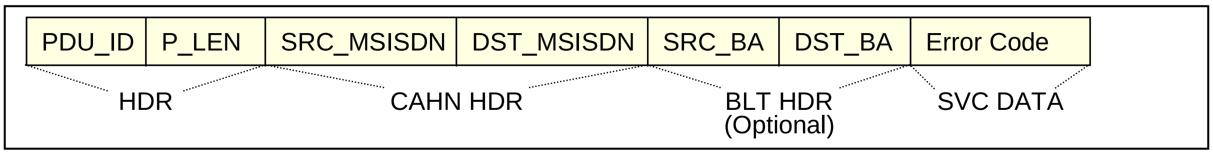


Figure 5.3 CAHN Error Response

The header fields did not change compared to the previously defined messages. But errors in CAHN are not always related to Bluetooth and therefore the Bluetooth header is optional in this kind of message. The service data (name is not very appropriate) contains the code of the error that has been detected. In case where the application is not able to assign a specific error code to an error, a standard error code is used. The definitions of these codes will follow later in this document. For the sake of completeness here the added field:

- Error code contains a predefined code for indication of a certain error. If the occurred error does not belong to a predefined code, a standard code is used instead.

5.4 CAHN CORE

Summarizing the requirements specification, the CAHN core must be an application, which implements the server part of CAHN as well as the client part. A server typically is a service, which is running in the background of the operating system waiting for requests. The server then handles the request and can perform actions to fulfill a request. When needed, a server can also give back a response. A client on the other hand is an application, which contacts a server to send a request to. When needed the client can get a response and handle this response. In the case of CAHN the application has to provide both of these entities, the client and the server. Which role the application will execute depends on the situation.

As the application must provide a CAHN server implementation the design suggests to make the CAHN application running in the background, waiting for requests. As on every CAHN enabled device thus a server is running, the client part of the CAHN application is able to contact the local server to hand over requests. Whether the request has to be handled locally or on a remote device will be decided by the server, which therefore must be able to either handle the request locally or to relay the request to a remote server, which can handle the request. When a request is relayed to a remote device, the local CAHN server plays the role of a client, and must therefore provide a mean for receiving responses, which have to be handed back to the request originator. The state diagram presented in Figure 5.5 illustrates the different states of the CAHN application.

In this sketch three different states are presented which divide the application in three logical parts:

1. The CAHN Communication Manager (waits for and handles requests)

2. The CAHN Connector (manages the access technology related issues)

3. The CAHN Adapter (provides an interface to the used cellular network for the CCM)

Figure 5.4 shows the different Core components. In the following chapters the components are explained to justify the chosen design.

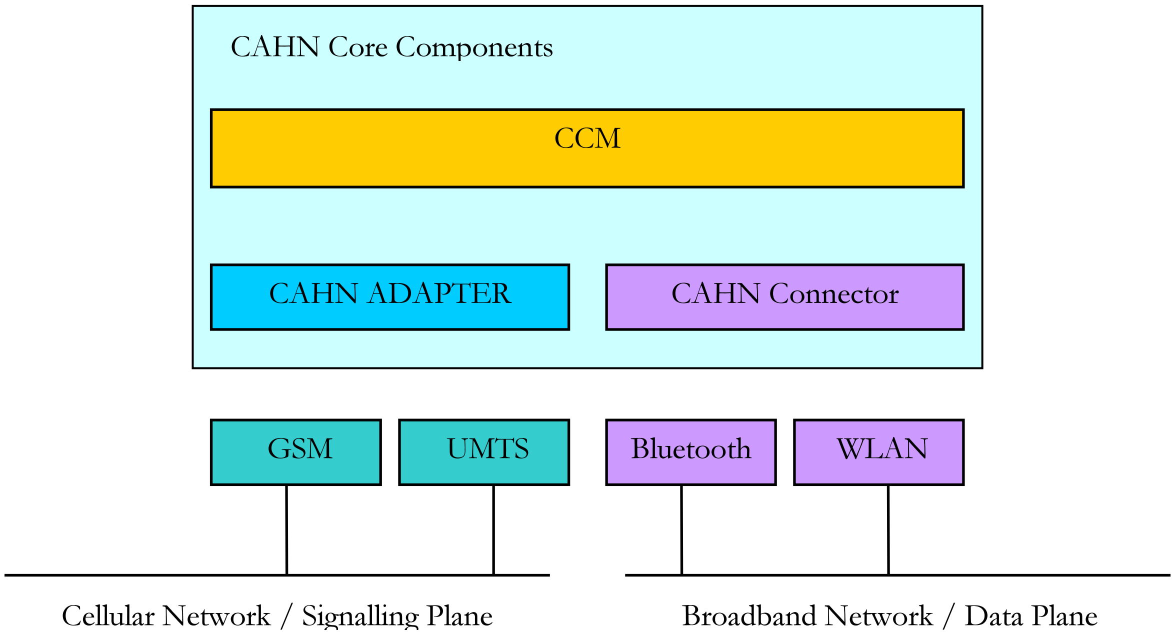


Figure 5.4 : The CAHN Core components

5.4.1 CAHN COMMUNICATION MANAGER (CCM)

This part of the application is responsible to wait for incoming requests, to handle those accurately and to provide a response. A request will be analyzed and the decision is made, whether the request has to be handled locally or relayed to a remote device. If the request has to be handled locally, it is further analyzed, to which access technology the request belongs to. To do so, this component must also provide the CAHN protocol definition, as the PDU_ID is used to identify the packet type. As the request handling procedure can depend on the access technology concerned in the request, a separate component for each access technology is proposed for the request handling. This design makes it possible to easily add new components for additional access technologies. This component is called the CAHN connector.

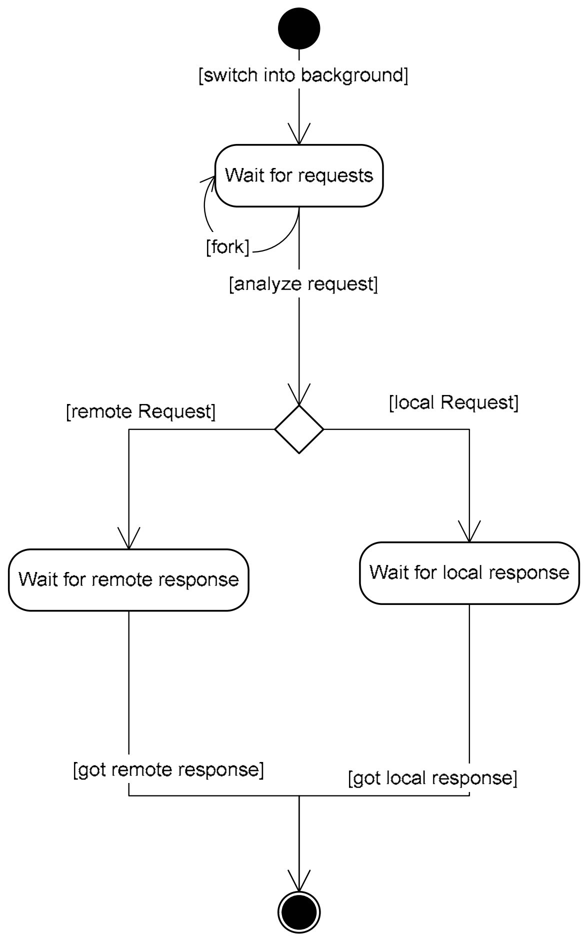


Figure 5.5: State Diagram for the CAHN protocol implementation

5.4.2 CAHN CONNECTOR

As mentioned before, the CAHN connector is responsible to handle requests belonging to the access technology it is responsible for. For each access technology a separate CAHN connector is used. Besides this the CAHN connector has to manage the access interface. This means, that it is responsible to setup the access interface according to the request. After the completion of the request handling, the CAHN connector returns a response to the CCM which will be responsible to delegate this response to the request initiator.

In the case of this diploma work, only a CAHN connector for the Bluetooth technology is defined. Figure 5.6 shows the actions a CAHN connector for Bluetooth has to perform to handle a request.

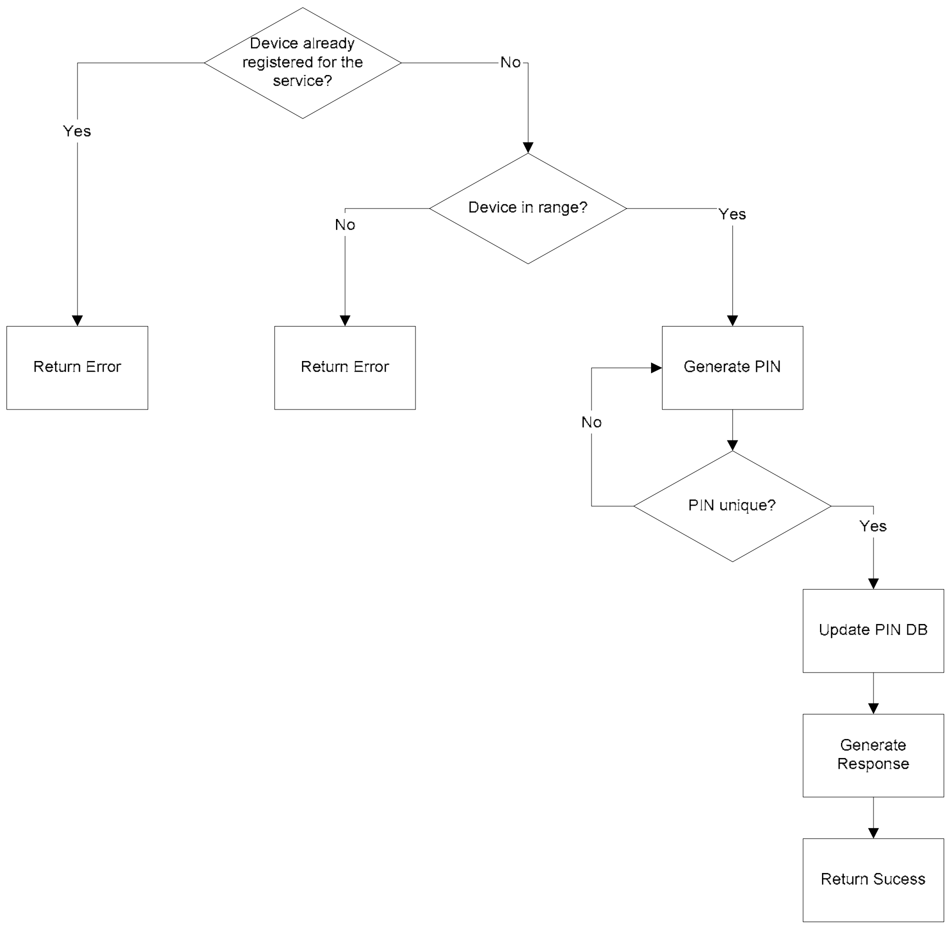


Figure 5.6: Flow chart of the CAHN connector for Bluetooth

5.4.3 CAHN ADAPTER

If a request is handed to the CAHN adapter the request must be handled remotely and thus forwarded over the cellular network. The CAHN adapter is responsible to take CAHN requests, transform them into cellular messages and send these cellular messages over the cellular network. When the response has been generated and arrives from the remote host, the CAHN adapter has to convert the cellular response message back into a CAHN response and to give it over to the CCM. The CAHN adapter can also get requests over the cellular network and must does be running continuously to be ready to receive remote requests. Remote cellular request messages are converted in CAHN request and handed to the CCM. The CCM will then give back a response to the CAHN adapter which will convert the response into a cellular message and send it back to the request initiating node.

In order to provide a standard interface to the cellular network for the CCM, the CAHN adapter provides a socket interface. The CCM uses that interface, when a request has to be relayed to a remote interface. The CAHN adapter converts the message, sends it over the cellular network, waits for the response, and gives this response to the CCM. Figure 5.7 below illustrates the message flow between the CCM and the CAHN adapter for the relaying.

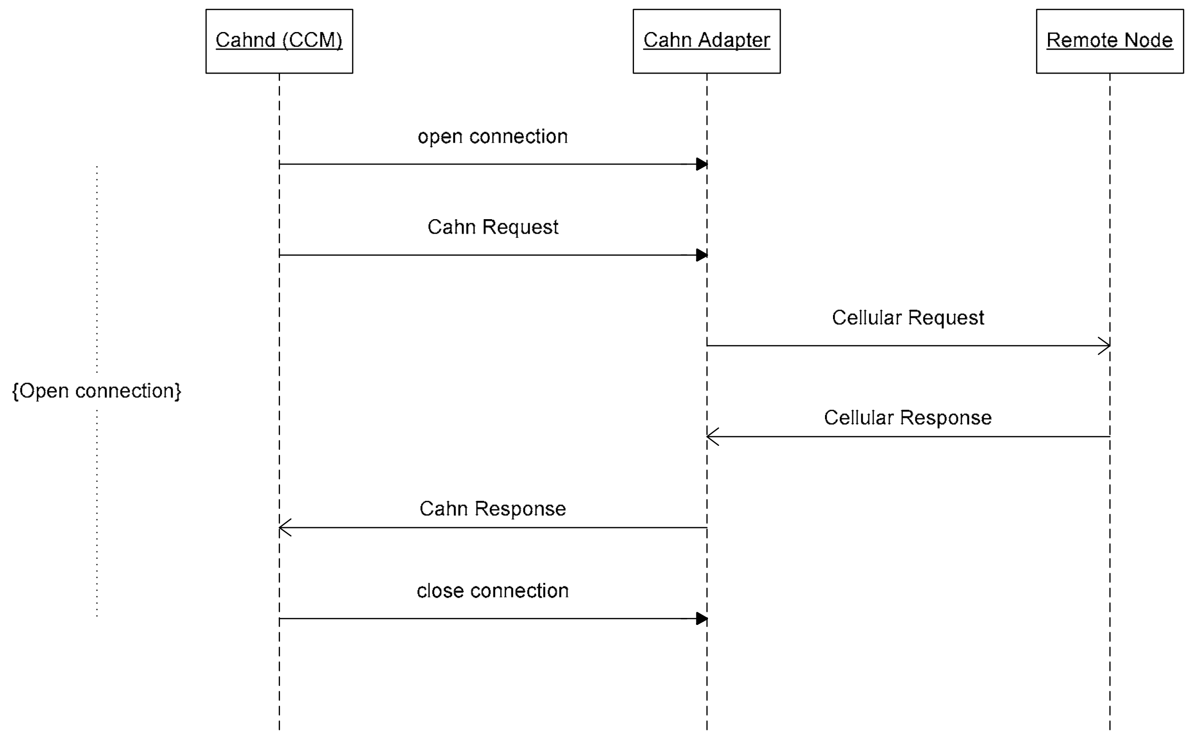


Figure 5.7: Relaying of a remote request

The problem with this approach is that the connection between the CCM and the CAHN adapter is open until the response arrives. This blocks the CCM in the "wait for response" state. Therefore the CCM will not be able to treat further requests, until a response arrives. To prevent this blocking of the CCM the connection must be torn down when the request is sent, and re-established, when the response arrives. Then the CCM would not be blocked anymore. This is presented in figure 5.8.

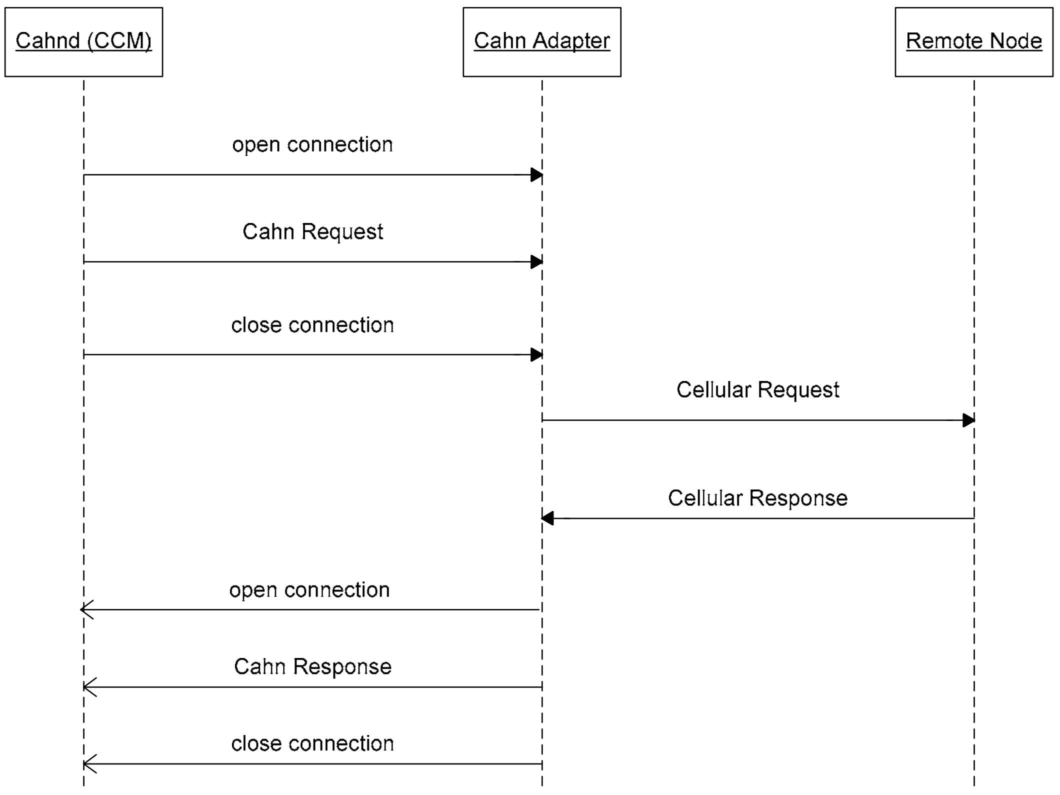


Figure 5.8: Relaying of a remote request without blocking cahnd

This indeed prevents the CCM from blocking, but with this solution, the context between the connection and the response will be lost. When the CAHN adapter gets the response, it can establish a connection to the CCM and hand over the response, but the CCM will not be able to distinguish to which request the response belongs to. Therefore every request that is sent from the CCM to the CAHN adapter is subsequently numbered. This sequence number is given to the adapter, who appends it to the packet, and is thus sent to the remote host, who includes it in the response. Once the response arrives at the CAHN adapter, the context to the request can be re-established. To send responses to the CCM the CAHN adapter has to use an interface, that is provided by the CCM. The UNIX socket interface that has been created on the CCM for incoming requests can not be used, as it was strictly defined as an interface for requests. Therefore another interface for incoming responses has to be created by the CCM. To provide those interface, the CCM create for each outgoing request a UNIX socket interface, with the name of the sequence number. So the CAHN adapter can connect to the socket indicated by the sequence number of the received response, and send the response to the CCM. The CCM can now handle that packet and send back a response to the request initiator. Like this, the problem of the blocking CCM can be solved, without losing the context between a certain request and its appropriate response.

This issue was rather related to the design of the CCM than to the implementation of the CAHN adapter. But those design decisions also have an impact on the design of the CAHN adapter and were therefore presented in this chapter. The fact that the CCM does not block and is thus able to handle different independent requests force the implementation of the CAHN adapter to supervise two interfaces at the same time: the interface for incoming CAHN messages originating from the CCM and the interface for incoming cellular messages. This is not possible in the present design, as cellular message reception is treated different from incoming packets over a socket interface. To cope with this issue, the CAHN adapter is split up in two parts, the CAHN adapter and the CAHN SMS adapter. The CAHN adapter is responsible for the surveillance of the UNIX socket interface provided for the CCM and the CAHN SMS adapter is responsible to handle incoming cellular messages. This has the advantage, that incoming cellular messages can be converted in the CAHN SMS adapter and sent to the same socket interface of the CAHN adapter that is provided for the CCM. So the CAHN adapter is only responsible for one interface, which is a UNIX socket interface.

5.4.4 CAHN SMS ADAPTER

The CAHN SMS adapter is a single stand-alone program that is invoked by the upon cellular message reception. The cellular message is received as a parameter by the CAHN SMS adapter, which first checks, whether the cellular message is a CAHN message. If not, the CAHN SMS adapter writes the message back on the cellular device. If the message is CAHN related, the CAHN SMS adapter first checks what kind of message it is and treats it accordingly. If the message is a request, the CAHN SMS adapter converts the message in a CAHN Bluetooth Service Request and sends it to the CAHN Adapter. If the message is an error message or a response, the message is converted into a CAHN Error respectively into a CAHN Bluetooth Service Response and sent to the appropriate response socket of the CCM. Figure 5.9 illustrates the performed actions in a flow chart.

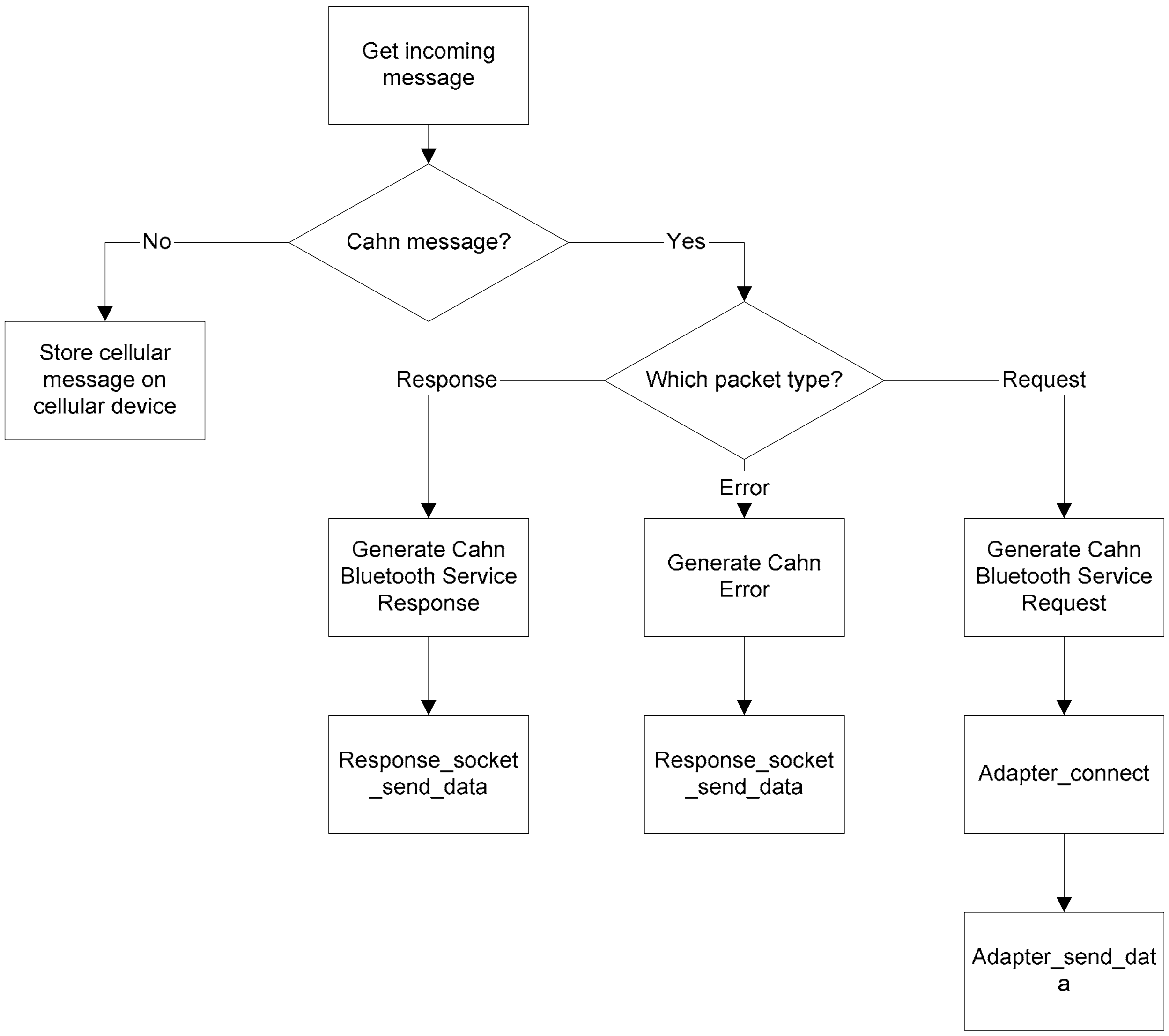

Get incoming message
Cahn message?
No
Yes
Store cellular message on cellular device
Which packet type?
Response
Request
Error
Generate Cahn Bluetooth Service Response
Generate Cahn Error
Generate Cahn Bluetooth Service Request
Response_socket _send_data
Response_socket _send_data
Adapter_connect
Adapter_send_dat a

Figure 5.9: Cellular message processing in CAHN SMS adapter

5.5 THE GUI

For the user interaction with the tool, a GUI has to be provided. As the CCM does offer a standard socket interface, the GUI can reuse this interface for GUI interaction with the CAHN core components. Therefore no further programming interface for possible user interfaces is provided. More details on the implemented GUI will follow in the implementation part of this document.

5.6 MESSAGE FLOW

To present the message flow among the different components, a diagram is provided. This diagram is split in to parts, the client and the server.

5.6.1 CLIENT SIDE

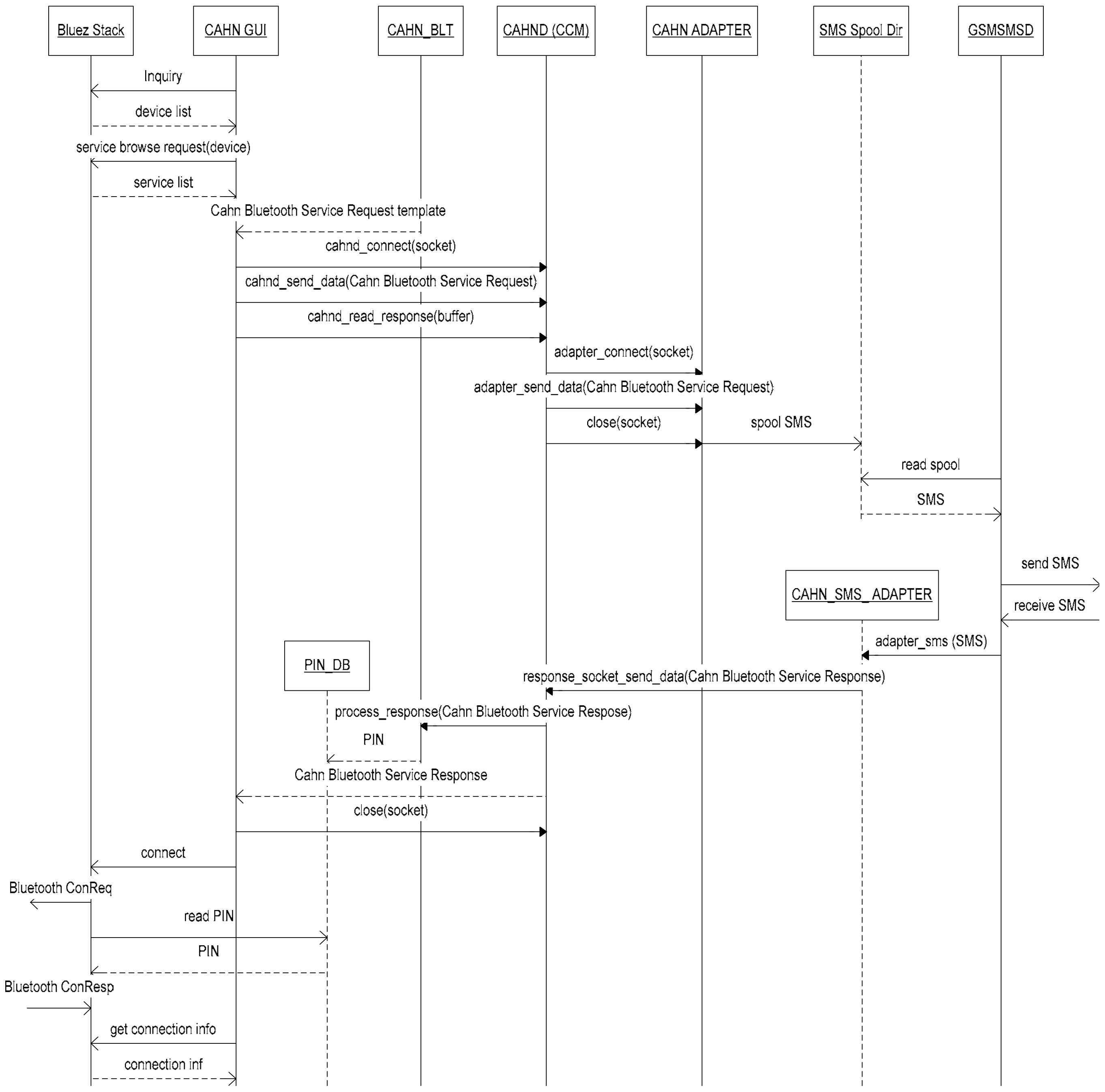


Figure 5.10: Message flow on client side

5.6.2 SERVER SIDE

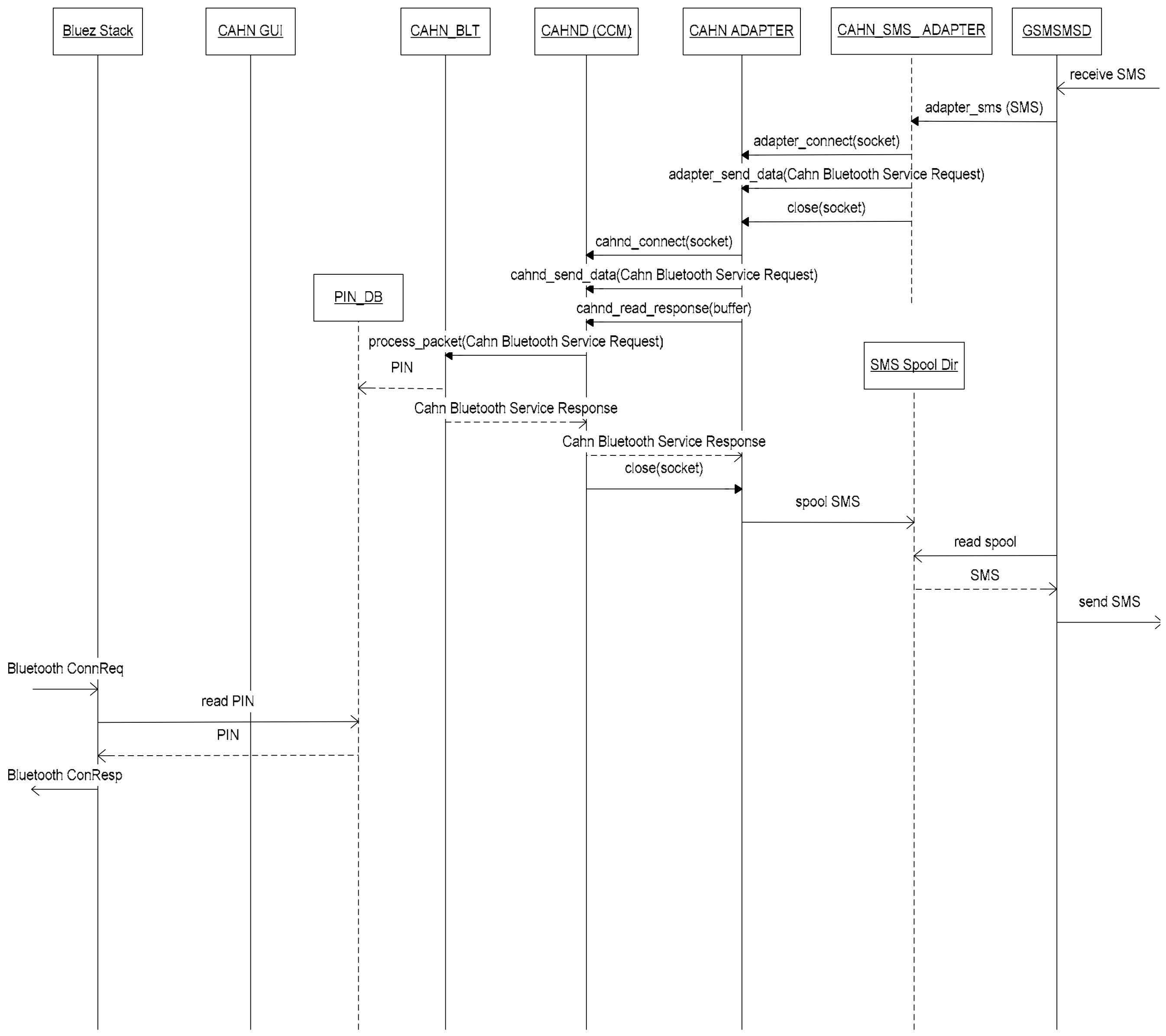


Figure 5.11: Message flow on server side

6 IMPLEMENTATION

6.1 INTEGRATION OF THE CAHN BLUETOOTH SERVICE AND THE CAHN BLUETOOTH SERVICE ATTRIBUTE IN THE BLUETOOTH SDP IMPLEMENTATION

6.1.1 USED BLUETOOTH IMPLEMENTATION

The platform, where the implementation is developed is a GNU/Linux system. The operating system is Red Hat 9.0. This choice has also influenced the decision of the Bluetooth stack that is used for this implementation. The Bluetooth stack implementation providing the needed hardware drivers, the necessary protocols and the profile implementations to realize the application is the Bluez stack [26]. Bluez was initially developed at Qualcomm but has moved to Open Source. Since kernel version 2.4.6 of GNU/Linux the Bluez device drivers are a fixed component of the kernel. This made Bluez become the de-facto standard Bluetooth stack of GNU/Linux and therefore the stack is integrated into the Red Hat 9.0 distribution. Besides the device drivers, the basic layers, namely the Bluetooth baseband, HCI and L2CAP are integrated in kernel space. Further the Bluez protocol stack provides user space implementations of different Bluetooth profiles and command-line tools for user interaction with the stack. Amongst others, Bluez offers implementations of the Bluetooth SDP and the PAN profile. And last but not least the Bluez protocol stack provides standard UNIX socket interfaces to all the implemented Bluetooth stack layers.

Figure 6.1 shows a detected NAP service represented on a Bluez client. The Service Record Handle (AttributeID: 0x0) of the NAP service is in this example 0x804ccf0. The human readable representation in the picture has been added by the Bluetooth SDP client and is not exchanged within Bluetooth SDP messages. The Service Class ID List (AttributeID: 0x1) contains only one Service Class with the ID 0x1116, which is the ID for the NAP Service Class. The Protocol Descriptor List (AttributeID: 0x4) contains two protocols, which are needed for the NAP service, namely L2CAP (UUID: 0x0100) and BNEP (UUID: 0x000f). Additional information on the specific use of the protocols can also be added, i.e. the used port number for an IP service. The Browse Group List (AttributeID: 0x5) is used to configure the reactions of the Bluetooth SDP server on service browse requests and service search operations and is not interesting for our purpose. The Bluetooth Profile Descriptor List (AttributeID: 0x9) contains a list of the attributes and their respective values, which have additionally defined for that service. In this case it only contains the UUID that has been defined for the NAP service. Finally the Human Readable String (AttributeID: 0x100) contains the human readable description of the given service.

```
Attribute Identifier : 0x0 - ServiceRecordHandle
  Integer : 0x804ccf0
Attribute Identifier : 0x1 - ServiceClassIDList
  Data Sequence
    UUID16 : 0x1116 - NAP (PAN/BNEP)
Attribute Identifier : 0x4 - ProtocolDescriptorList
  Data Sequence
    Data Sequence
      UUID16 : 0x0100 - L2CAP
      Channel/Port (Integer) : 0xf
    Data Sequence
      UUID16 : 0x000f - BNEP (PAN/BNEP)
      Channel/Port (Integer) : 0x100
      Data Sequence
        Protocol (Integer) : 0x10
        Channel/Port (Integer) : 0x20
        Version (Integer) : 0x30
        Integer : 0x40
Attribute Identifier : 0x5 - BrowseGroupList
  Data Sequence
    UUID16 : 0x1002 - PublicBrowseGroup (SDP)
Attribute Identifier : 0x9 - BluetoothProfileDescriptorList
  Data Sequence
    Data Sequence
      UUID16 : 0x1116 - NAP (PAN/BNEP)
      Version (Integer) : 0x100
Attribute Identifier : 0x100
  Text : "Network Access Point Service"
```

Figure 6.1: Detected NAP service

6.1.2 BLUETOOTH SDP MODIFICATIONS

To integrate the defined CAHN Bluetooth service and the CAHN Bluetooth service attribute, modifications to the existing implementation of the Bluetooth SDP provided by the Bluez Bluetooth stack have to be made. For that purpose the relevant parts of the implementation are identified:

- sdp.h
  In this file the IDs of the Services Classes, the Bluetooth profiles and of the Attributes are allocated. The adoptions are outlined in red color in figure 6.2

```
[..]
/*
 * Service class identifiers of standard services and service
groups
 */
#define SDP_SERVER_SVCLASS_ID          0x1000
#define BROWSE_GRP_DESC_SVCLASS_ID     0x1001

[..]

#define GENERIC_AUDIO_SVCLASS_ID            0x1203
#define GENERIC_TELEPHONY_SVCLASS_ID        0x1204

// Added for CAHN Service Class ID
#define CAHN_SVCLASS_ID                0x1220

/*
 * Standard profile descriptor identifiers; note these
 * may be identical to some of the service classes defined
above
 */
#define SERIAL_PORT_PROFILE_ID         0x1101
#define LAN_ACCESS_PROFILE_ID               0x1102

[..]

#define HID_PROFILE_ID                      0x1124
#define CIP_PROFILE_ID                      0x1128

// Added for CAHN Profile ID
#define CAHN_PROFILE_ID                     0x1220

/*
 * Possible values for attribute-id are listed below.
 * See SDP Spec, section "Service Attribute Definitions" for
more details.
 */
#define SDP_ATTR_RECORD_HANDLE         0x0000

[..]

#define SDP_ATTR_IP6_SUBNET            0x030E

// Added for CAHN MSISDN attribute and CAHN service attribute
#define SDP_ATTR_CAHN_MSISDN           0x0400
#define SDP_ATTR_CAHN_ENABLED_SERVICE    0x0401

[..]
```

Figure 6.2: Modifications to sdp.h

- uuid.c

  As seen in Figure 6.1, the Bluez Bluetooth SDP client implementation adds human readable descriptions to Service Class IDs and the Profile ID. This description is defined in uuid.c. Changes to this file are highlighted in red again.

```
[..]

static struct tupla ServiceClass[] = {
      { SDP_SERVER_SVCLASS_ID,          "SDP Server" },
      { BROWSE_GRP_DESC_SVCLASS_ID,     "Browse Group Descriptor" },

[..]

      { GN_SVCLASS_ID,                  "PAN group network" },
      { CAHN_SVCLASS_ID,                "CAHN enabled device" },
      { 0 }
};

static struct tupla Profile[] = {
      { SERIAL_PORT_PROFILE_ID,          "Serial Port" },
      { LAN_ACCESS_PROFILE_ID,           "LAN Access Using PPP" },

[..]

      { NAP_PROFILE_ID,                  "PAN access point" },
      { GN_PROFILE_ID,                   "PAN group network" },
      { CAHN_PROFILE_ID,                 "CAHN enabled device" },
      { 0 }
};

[..]
```

Figure 6.3: Modifications to uuid.c

- sdptool.c and listattr.c

  These files are part of a command line tool to perform Bluetooth SDP operations. It does not directly influence the functionality of the Bluetooth SDP and is therefore not necessary to be changed. Nevertheless the tool was modified to provide the operations also for the CAHN service and the CAHN attribute. But the changes will not be further mentioned hereafter. The changed code is supplied as add-on.

With the help of the previous modifications to the different files of the bluez_sdp package, the CAHN service and the CAHN service attribute can now be added as a Service Record respectively as an additional service attribute to an existing Service Record to the Bluetooth SDP server. Like this the device can indicate the CAHN capability and the Bluetooth services that are enabled for CAHN. It is important to get straight that these same modifications have to done on every client device to recognize the new service and the new service attribute. To deal with this a profile must be defined, which has to be standardized through the Bluetooth SIG. Once the profile makes part of the Bluetooth standard, it is likely to be integrated in future implementations and would make changes to the source code unnecessary. Figure 6.4 and figure 6.5 illustrate a detected CAHN service respectively a NAP service enabled for CAHN.

```
Attribute Identifier : 0x0 - ServiceRecordHandle
  Integer : 0x804d250
Attribute Identifier : 0x1 - ServiceClassIDList
  Data Sequence
    UUID16 : 0x1220 - CAHN enabled Device
Attribute Identifier : 0x5 - BrowseGroupList
  Data Sequence
    UUID16 : 0x1002 - PublicBrowseGroup (SDP)
Attribute Identifier : 0x9 - BluetoothProfileDescriptorList
  Data Sequence
    Data Sequence
      UUID16 : 0x1220 - CAHN enabled Device
      Version (Integer) : 0x100
Attribute Identifier : 0x100
  Text : "CAHN enabled device"
```

Figure 6.4: CAHN service detected

```
Attribute Identifier : 0x0 - ServiceRecordHandle
  Integer : 0x804ccf0

[..]

Attribute Identifier : 0x100
  Text : "Network Access Point Service"
Attribute Identifier : 0x401
  Text : "CahnEnabledService:0x804ccf0"
```

Figure 6.5: CAHN enabled NAP service detected

6.2 CAHN COMMUNICATION MANAGER (CCM)

The CCM is implemented in C. As the GNU/Linux kernel is implemented in C and also the Bluez Bluetooth stack, the choice of the C programming language will guarantee a high degree of interoperability. For the same reason the CAHN connector, the CAHN adapter and the CAHN SMS adapter are also implemented in C.

The CCM is implemented as a standalone program, which switches into background and waits for connections. As interface for connections a local UNIX socket interface has been chosen. The name of the application is cahnd which corresponds to the de-facto standard naming scheme used in GNU/Linux. The program is started with the command:

```
Linux# cahnd
```

and needs no further command line options. As the program is running in the background its output is send to the system log. Programs intending to interact with cahnd need to include the header file "cahnd_global_functions.h".

To establish a connection with cahnd, this header file provides a global function:

```
int cahnd connect()
```

This function can be called by clients and returns the number of the socket. In case of an error, the function's return value is -1. If no error occurred, the socket number can be used to send data to the CCM. This can be achieved with the global function:

```
int cahnd send data(int sock, char* buf, int size)
```

Here "sock" is the number of the socket, "buf" a character pointer to the data that has to be transferred and "size" indicates the length of the data transmission. The function returns 1, when sending was successful and -1, in case of errors.

After the successful transmission of data, which will be a CAHN request message, the client has interest to receive a response. It can read the response from the CCM with help of the global routine:

```
int cahnd read response(int sock, char *rsp buf, int rsp buf len)
```

The parameter "sock" again is the socket number. "rsp_buf" is a character pointer, which indicates the beginning of a previously allocated memory area, where the response will be saved to. After the execution of the function "rsp_buf_len" will hold the length of the response packet.

As explained in the design, cahnd must provide response sockets, where the adapter can send the responses to maintain the request-response context. These socket interfaces are also local UNIX sockets. The adapter can connect and send responses to the CCM with help of:

```
int  response socket send data(char*  data,  int  data length,  int
response _socket)
```

In this function "data" is the character pointer to the buffer holding the response, "data_length" represents the size of the response, and "response_socket" is the number of the response socket, where the response will be sent to.

Besides the global functions, which can be used for the interaction with cahnd, the implementation of the CCM contains global CAHN definitions for the CAHN protocol. The following lines define the PDU_IDs for the different CAHN protocol messages:

```
/*
 * The PDU identifiers of CAHN packets between client and server
 */
#define CAHN_ERROR_RSP        0x01
#define CAHN_BLT_SVC_REQ      0x02
#define CAHN_BLT_SVC_RSP      0x03
```

The lines below specify the packet header

```
/*
 * CAHN PDU
 */
typedef struct {
        uint8_t   pdu_id;
        uint16_t  plen;
} __attribute__ ((packed)) cahn_pdu_hdr_t;
```

and those ones the CAHN header:

```
/*
 * CAHN MSISDN HEADER
 */
typedef struct {
        char src_msisdn[MAX_SIZE_OF_MSISDN];
        char dst_msisdn[MAX_SIZE_OF_MSISDN];
} cahn_msisdn_hdr_t;
```

The messages that are defined for the CAHN protocol concern all Bluetooth related services. Therefore the definitions of the entire CAHN Bluetooth Service Request and CAHN Bluetooth Service Response messages will follow in the implementation of the CAHN connector. Error messages are not necessarily Bluetooth related, and the use of the Bluetooth header is optional. Thus error messages are also defined here:

```
/*
 * ERROR Packet
 */
typedef struct {
        char src_msisdn[MAX_SIZE_OF_MSISDN];
        char dst_msisdn[MAX_SIZE_OF_MSISDN];
        int error_code;
} cahn_error_packet_t;
```

In the definition of these error messages an Integer value called "error_code" gives a reference to the error that occurred. These error codes were assigned as follows:

```
/*
 * Error codes
 */
#define CAHN_UNKNOWN_PDU_ID                     0x01
#define CAHN_INVALID_PDU_SIZE                   0x02
#define CAHN_CONNECT_ERROR                      0x03
#define CAHN_SEND_ERROR                         0x04
#define CAHN_READ_ERROR                         0x05
#define CAHN_INTERNAL_SERVER_ERROR              0x06
#define CAHN_DEVICE_NOT_IN_RANGE_ERROR          0x07
#define CAHN_DEVICE_ALREADY_REGISTERED_ERROR    0x08
#define CAHN_ADAPTER_SEND_ERROR                 0x09
#define CAHN_ADAPTER_READ_ERROR                 0x10
#define CAHN_PROCESS_RSP_ERROR                  0x11
#define CAHN_PREPARE_RESPONSE_SOCKET_ERROR      0x12
```

Up to now there are definitions for the global CAHN protocol message header with the PDU_ID and the PLEN fields, with the definitions of the PDU_IDs; for the CAHN header with the SRC_MSISDN and the DST_MSISDN fields; and for the error messages not related to Bluetooth, with appropriate error codes. What misses now to complete the definition of the CAHN protocol are the Bluetooth related parts. These will be topic of the next chapter.

6.3 CAHN CONNECTOR

To complete the description of the CAHN protocol messages, this section about the implementation of the CAHN connector will start with the CAHN protocol related definitions. The following struct definitions are used to complete the CAHN Bluetooth Service Request and the CAHN Bluetooth Service Response message. They make part of cahn_blt.h:

```
/*
 * CAHN Service Request Record
 */
typedef struct {
  int       service_ID;
  bdaddr_t  src_bdaddr;
  bdaddr_t  dst_bdaddr;
} cahn_blt_svc_req_rec_t;
```

```
/*
 * CAHN Service Response Record
 */
typedef struct {
  int       service_ID;
  bdaddr_t  src_bdaddr;
  bdaddr_t  dst_bdaddr;
  int       pin;
} cahn_blt_svc_rsp_rec_t;
```

This concludes now the protocol definition. For the sake of completeness the Bluetooth related error codes are listed, too:

```
/*
 * CAHN BLT Error Codes
 */
#define CAHN_BLT_PIN_DB_CONNECT_ERROR       0x01
#define CAHN_BLT_PIN_DB_QUERY_ERROR         0x02
#define CAHN_BLT_DEVICE_NOT_IN_RANGE_ERROR  0x03
#define CAHN_BLT_ALREADY_REGISTERED_ERROR   0x04
#define CAHN_BLT_SVC_RSP_REC_PARSE_ERROR    0x05
```

As mentioned in the design, the responsibility to handle a request, which is related to Bluetooth, has been moved over to the CAHN connector. The responsible request handler function is coded in the file cahn_blt.c and looks like this:

```
int blt process packet(cahn blt svc req rec t *req, char *rsp buf)
```

"req" points to the place, where the request is stored in memory, and "rsp_buf" points to a pre-allocated memory space which will hold the response generated by the handler function. The function's return type is 0 in case of success or is assigned the respective error code (which was just presented) in case of an error.

The different actions, the function has to perform to handle a Bluetooth Service Request and to generate an appropriate Bluetooth Service Response message are illustrated in the flow chart in figure 5.6

For each of the actions shown in figure 5.6 a help function has been implemented in cahn_blt.c. As these functions are private they are not interesting for any interaction. Therefore, they are just listed, but not further explained. The interested reader may have a look at the source code.

- `int device_service_registered(bdaddr_t *bdaddr, int service)`
  To check whether a device has already been registered for the service in the PIN database.

- `int device_in_range(bdaddr_t *bdaddr_local, bdaddr_t *bdaddr_remote)`
  To check whether the device from which the request originates is in the range of Bluetooth radio.

- `int generate_pin()`
  To generate a PIN.

- `int test_pin(int pin)`
  To check whether the PIN does not already exist in the database. This is to prevent multiple usage of the same PIN.

- `int add_rec_to_pinDB(cahn_blt_svc_req_rec_t *rec, int pin)`
  To add a new record to the PIN database. This function implements the interface to the PIN database.

The CAHN connecter does not only have to handle local CAHN Bluetooth Requests, but also remote CAHN Bluetooth Responses. When the CCM receives an answer which would result in Bluetooth related actions, like storing the Pin obtained by a peer, it will relay the response to the CAHN connector to perform the needed tasks. The function to handle these responses is called

```
int blt process response(char *rsp buf, int rsp buf len)
```

"rsp" points to the Response, that has to be treated and "rsp_buf_len" contains the length of this response. This function returns 0 in case of success or the respective error code in case of an error. The only responsibility of this function is to store the obtained PIN in the PIN database. If already a record for this service is present on the PIN database, it will be overwritten.

Not all requests arriving to the CCM are intended to be handled locally and must therefore be relayed to the appropriate client. This relaying is done in the CAHN adapter, which's implementation is topic of the following chapter.

6.4 CAHN ADAPTER

The main task of the CAHN adapter is to relay packets that arrive from the CCM to the GSM network. To do so, the CAHN adapter provides a local UNIX socket interface. The CCM connects to that interface and hands the packet to the CAHN adapter. The CAHN adapter converts the packet into an SMS messages and sends it over the GSM network to the receiver MSISDN. If the packet is a request, the receiving peer will generate a response, which is then sent back over the GSM network to the CAHN adapter. The CAHN adapter converts the SMS message into a CAHN message and hands it to the CCM. Figures 5.7 and 5.8 illustrate the process of relaying a request to a remote node. The CAHN adapter also has the purpose to provide a standardized interface to the cellular network. In the design was defined, that this interface is a standard UNIX socket and the cellular network is the GSM network.

To use this socket interface, the CAHN adapter provides global functions, which are implemented in cahn_adapter_global_functions.h:

```
int adapter connect()
```

This function is used by the CCM and the CAHN SMS adapter to connect to the CAHN adapter socket interface. The return value is the number of the socket, or a negative value in case of an error. To send data to that socket, the following function is implemented,

```
int  adapter send data(int  sock,  char  *buf,  int  size,  int
resp_sock)
```

where "sock" contains the number of the socket data is sent to, "buf" points to the location, where the data is stored and "size" indicates the amount of data that is sent. The parameter "resp_sock" now is used, to get the sequence number of a packet from the CCM.

Remember that a function is provided by the implementation of the CCM (cahnd) for the CAHN adapter to send a response to a response socket with the name of the sequence number:

```
int  response socket send data(char*  data,  int  data length,  int
response _socket)
```

This function has already been explained in the section about the CCM.

6.5 CELLULAR NETWORK

The attentive reader may already have recognized from the nomenclature in the previous chapter that the cellular network of choice is the GSM network. The GSM network is in a productive state at the moment and can therefore be used for testing. As transport mechanism for cellular messages the Short Message System (SMS) is used. The choice of SMS is reasonable as SMS can be used without a further setup with the GSM operator.

To access the GSM network, ordinary GSM cell phones are chosen. They are connected either with a serial cable or with help of a Bluetooth link. For the interaction with the phones, a third party software named gsmlib is employed.

6.5.1 GSMLIB

On the homepage of the gsmlib the following description of the tool is available:

„This distribution contains a library to access GSM mobile phones through GSM modems. Features include:

- modification of phonebooks stored in the mobile phone or on the SIM card
- reading and writing of SMS messages stored in the mobile phone
- sending and reception of SMS messages

Additionally, some simple command line programs are provided to use these functionalities. "

No further details are presented here and just the use of a tool provided by gsmlib is explained. The tool is called gsmsmsd and is a daemon program. It runs in the background and monitors an attached GSM phone for incoming messages. Additionally the tool can check in a pre-defined spool directory for spooled SMS messages and send them. The tool is started with the command

```
Linux# gsmsmsd packet type
```

where packet type specifies what kind of SMS messages are monitored. Type can be a combination of the following

- sms, no_sms (for regular SMS messages)
- cb, no_cb (for Cell Broadcast messages)
- stat, no_stat (for Status messages)

The behaviour of the daemon can be controlled with help of the command parameters. The parameters of interest are:

- -a *command*
  Invokes the specified command upon SMS reception. In our case the daemon will invoke the CAHN SMS adapter

- --spool *spool_dir*
  The directory it checks for spooled SMS messages

- -L
  Messages are sent to syslog

- -S *outbox_dir*
  A directory where a copy of the sent SMS message can be stored. This is optional, but nice for debugging and cost control

- -d *dev*
  The device where the mobile is connected

So with this information the tool that monitors the SMS interface is already provided and it can be setup to invoke the CAHN SMS adapter upon SMS reception. The implementation of the gsmsmsd is already provided by the gsmlib program suite. For the sake of completeness the command which is used to invoke it:

```
Linux # gsmsmsd sms no_cb no_stat –a adapter_sms –spool
sms_spool –d device
```

6.6 CAHN SMS ADAPTER

As defined in the design, the CAHN SMS adapter is a standalone program, which is invoked upon packet receptions. As soon as an SMS arrives, the daemon, that is used to receive SMS, the gsmsmsd, will invoke the CAHN SMS adapter with the SMS as command argument:

```
Linux # adapter_sms sms_text
```

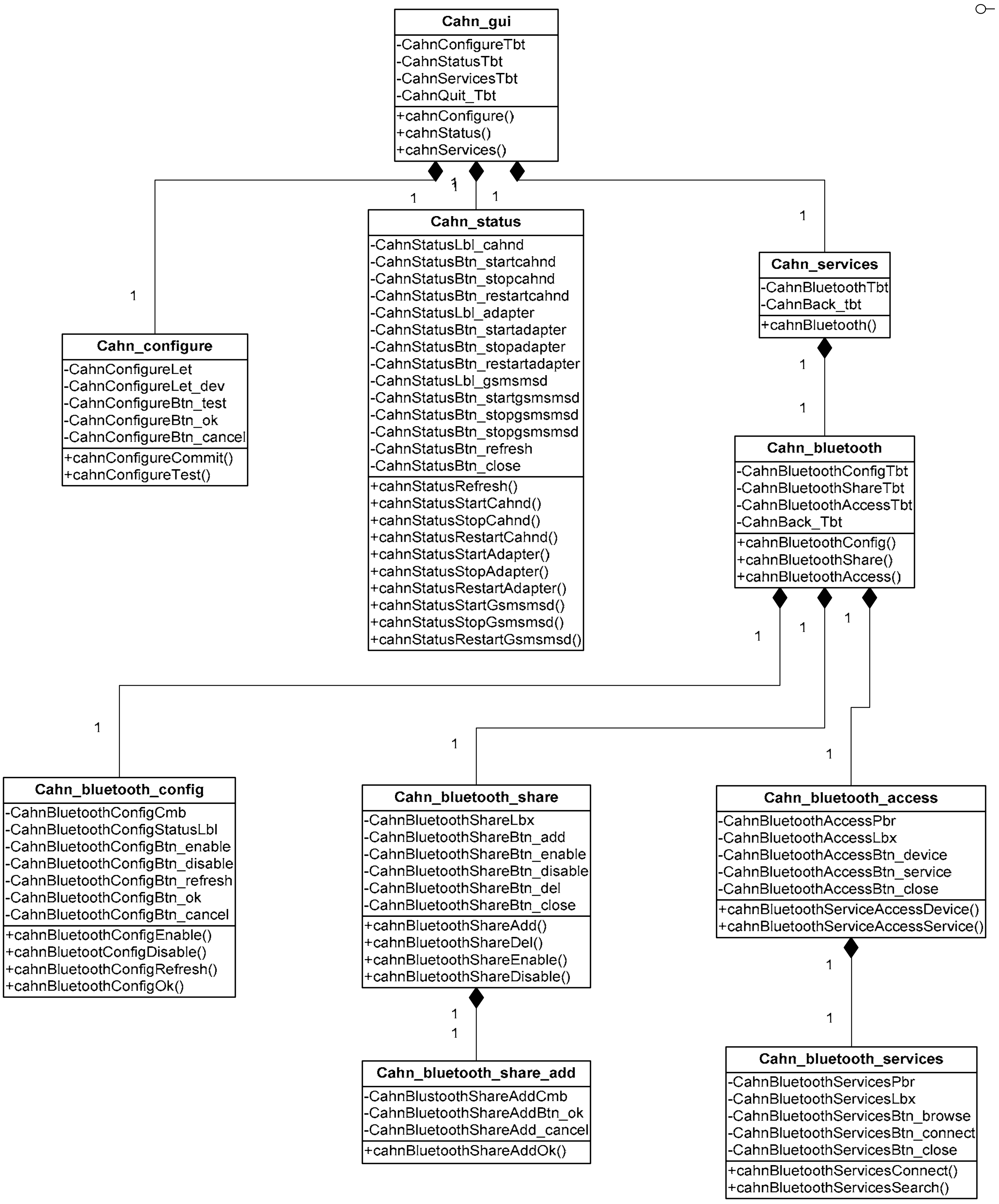
With the presented components, the CAHN protocol is entirely implemented. All mechanisms for CAHN protocol message treatment and delivery are now defined and implemented. But before CAHN protocol messages can move across a CAHN environment, they have to be created somehow. As CAHN is intended to act on behalf of the user, it is obvious, that a CAHN Service Request surely originates from the interaction with user. To provide this interaction between the user and the CAHN architecture a Graphical User Interface (GUI) has been implemented. This user interface serves as an example and was useful during the test and demonstration phase of the diploma work. But other user interfaces may be designed and used to control a CAHN implementation. The programming interface between the User Interface and the CCM is simply a local UNIX socket, where requests can be sent to. The following chapter will present the implementation of this GUI.

6.7 THE GUI

As the CCM does provide a standard UNIX socket interface for getting requests, the decision is made to reuse this interface for the GUI. So the GUI can be implemented in any programming language, which includes support for socket programming.

For that reason, it is proposed to use an object oriented approach. The GUI will be the interface for the user to control the implementation. Therefore the design must be intuitive and a window based approach is suggested. Every window treats a separate aspect of the implementation and for every window a class is modelled.

The main window, the user sees, must offer the three possibilities, to configure CAHN, to control the involved programs (the CAHN adapter, the CCM and the gsmsmsd) and to use the application. Therefore three sub-windows are implemented: a configuration window, a status window and a service window. In the configuration window, the user can enter the MSISDN of the attached mobile phone and the device address to communicate with the mobile phone. In the control window, the user can start/stop/restart the CCM, the CAHN adapter and gsmsmsd and in the service window the user can chose the type of service, he intends to use, i.e. Bluetooth or WLAN. In this implementation only the Bluetooth service is implemented. Once the user choses the Bluetooth service, he must be able to configure Bluetooth itself, the service sharing and he must be able to access to Bluetooth services. Therefore three other windows are used: the configure window, the sharing window and the access window. In the configure window, the user can chose the Bluetooth device (if more than one is present) he intends to use for CAHN. Further he can enable or disable the service sharing for Bluetooth. Enabling and disabling the service sharing means in the case of Bluetooth to start or stop the Bluetooth SDP implementation and to register and deregister the CAHN service in the Bluetooth SDP. In the service sharing window the user can chose to add a Bluetooth service to share (this means register it to the Bluetooth SDP) and enable or disable it for CAHN, what is nothing else, than to register or deregister the defined CAHN service attribute to the service. If the user wants to add a shared service, a window will pop up to give him the possibility to choose a service. In case of Bluetooth the access window is used to scan the environment for other devices. Once the scan is complete, a list of available devices will be show in the access window. Now the user can chose the device of interest and begin to scan for services. A window called service will come up, where the user can invoke a service scan to the chosen device. When the scan is completed, the user can chose a service and invoke the connection establishment to that service. This means, that a CAHN request is generated and sent to the CCM. Figure 6.6 contains the class diagram of the design of the GUI.



Cahn_gui
-CahnConfigureTbt
-CahnStatusTbt
-CahnServicesTbt
-CahnQuit_Tbt
+cahnConfigure()
+cahnStatus()
+cahnServices()
Cahn_configure
-CahnConfigureLet
-CahnConfigureLet_dev
-CahnConfigureBtn_test
-CahnConfigureBtn_ok
-CahnConfigureBtn_cancel
+cahnConfigureCommit()
+cahnConfigureTest()
Cahn_status
-CahnStatusLbl_cahnd
-CahnStatusBtn_startcahnd
-CahnStatusBtn_stopcahnd
-CahnStatusBtn_restartcahnd
-CahnStatusLbl_adapter
-CahnStatusBtn_startadapter
-CahnStatusBtn_stopadapter
-CahnStatusBtn_restartadapter
-CahnStatusLbl_gsmsmsd
-CahnStatusBtn_startgsmsmsd
-CahnStatusBtn_stopgsmsmsd
-CahnStatusBtn_stopgsmsmsd
-CahnStatusBtn_refresh
-CahnStatusBtn_close
+cahnStatusRefresh()
+cahnStatusStartCahnd()
+cahnStatusStopCahnd()
+cahnStatusRestartCahnd()
+cahnStatusStartAdapter()
+cahnStatusStopAdapter()
+cahnStatusRestartAdapter()
+cahnStatusStartGsmsmsd()
+cahnStatusStopGsmsmsd()
+cahnStatusRestartGsmsmsd()
Cahn_services
-CahnBluetoothTbt
-CahnBack_tbt
+cahnBluetooth()
Cahn_bluetooth
-CahnBluetoothConfigTbt
-CahnBluetoothShareTbt
-CahnBluetoothAccessTbt
-CahnBack_Tbt
+cahnBluetoothConfig()
+cahnBluetoothShare()
+cahnBluetoothAccess()
Cahn_bluetooth_config
-CahnBluetoothConfigCmb
-CahnBluetoothConfigStatusLbl
-CahnBluetoothConfigBtn_enable
-CahnBluetoothConfigBtn_disable
-CahnBluetoothConfigBtn_refresh
-CahnBluetoothConfigBtn_ok
-CahnBluetoothConfigBtn_cancel
+cahnBluetoothConfigEnable()
+cahnBluetootConfigDisable()
+cahnBluetoothConfigRefresh()
+cahnBluetoothConfigOk()
Cahn_bluetooth_share
-CahnBluetoothShareLbx
-CahnBluetoothShareBtn_add
-CahnBluetoothShareBtn_enable
-CahnBluetoothShareBtn_disable
-CahnBluetoothShareBtn_del
-CahnBluetoothShareBtn_close
+cahnBluetoothShareAdd()
+cahnBluetoothShareDel()
+cahnBluetoothShareEnable()
+cahnBluetoothShareDisable()
Cahn_bluetooth_access
-CahnBluetoothAccessPbr
-CahnBluetoothAccessLbx
-CahnBluetoothAccessBtn_device
-CahnBluetoothAccessBtn_service
-CahnBluetoothAccessBtn_close
+cahnBluetoothServiceAccessDevice()
+cahnBluetoothServiceAccessService()
Cahn_bluetooth_share_add
-CahnBlustoothShareAddCmb
-CahnBluetoothShareAddBtn_ok
-CahnBluetoothShareAdd_cancel
+cahnBluetoothShareAddOk()
Cahn_bluetooth_services
-CahnBluetoothServicesPbr
-CahnBluetoothServicesLbx
-CahnBluetoothServicesBtn_browse
-CahnBluetoothServicesBtn_connect
-CahnBluetoothServicesBtn_close
+cahnBluetoothServicesConnect()
+cahnBluetoothServicesSearch()
1

Figure 6.6: Class diagram of the CAHN GUI

As an object oriented approach for the implementation of the GUI is purposed, the GUI is implemented in C++. The choice of the C++ programming language made it possible to make use of QtDesigner, a tool to build graphical user interfaces. For each window that is built in QtDesinger the code for an abstract class is generated. This class includes the definitions of the methods which are called, when a GUI event (like pressing a button) occurs. These definitions contain no implementation. The classes are inherited from and the methods implemented, together with additional methods. For the sake of simplicity the abstract classes are not presented in the class diagram in figure 6.6

The Cahn_gui class

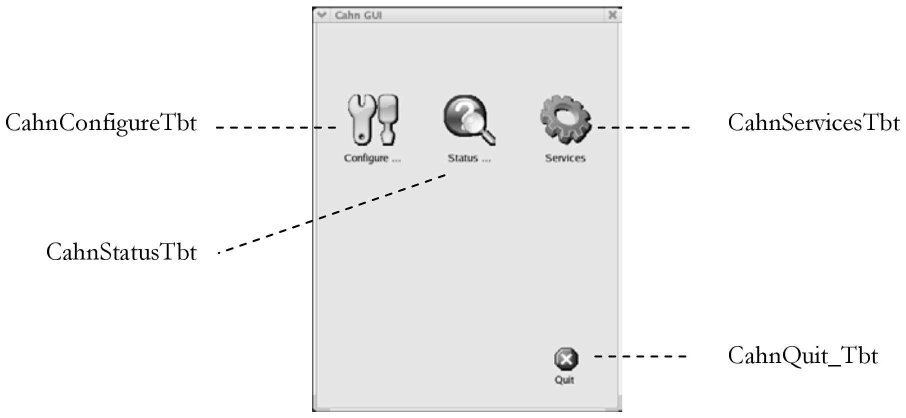


Figure 6.7: Screenshot Cahn_gui

When the GUI is started, an instance of the "Cahn_gui" class is created. This class is the main window of the GUI and provides three functions, which are invoked upon a key press event.

```
virtual void cahnConfigure()
```

is called when CahnConfigureTbt is pressed and creates an instance of Cahn_configuration which provides a window for the CAHN configuration.

```
virtual void cahnStatus()
```

is called when CahnStatusTbt is pressed and creates an instance of Cahn_status which provides a window to control the status of the involded daemons, namely cahnd, cahn_adapter and gsmsmsd.

```
virtual void cahnServices()
```

is called when CahnServiceTbt is pressed and creates an instance of Cahn_services, which provides a window to chose a service type.

If CahnQuit_tbt is pressed, a Qt internel function called accept() is executed, which closes the window.

The Cahn_configure class

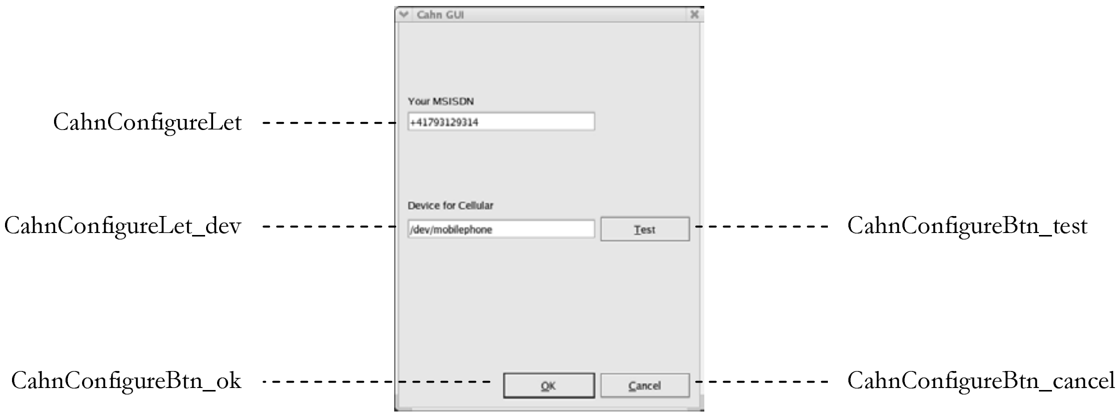


Figure 6.8: Cahn_configure screenshot

This window offers the user the possibility to configure CAHN. The values of interest are the MSISDN and the UNIX device representation (i.e. a serial port) to communicate with the attached GSM device.

```
virtual void cahnConfigureTest()
```

is called when CahnConfigureBtn_test is pressed and sends AT test commands to the specified attached GSM device for probing for its presence.

```
virtual void cahnConfigureCommit()
```

is called when CahnConfigureBtn_ok is pressed. It stores the specified MSISDN and the specified device to a file. This file is read by the CAHN application, which uses these settings.

The Cahn_status class

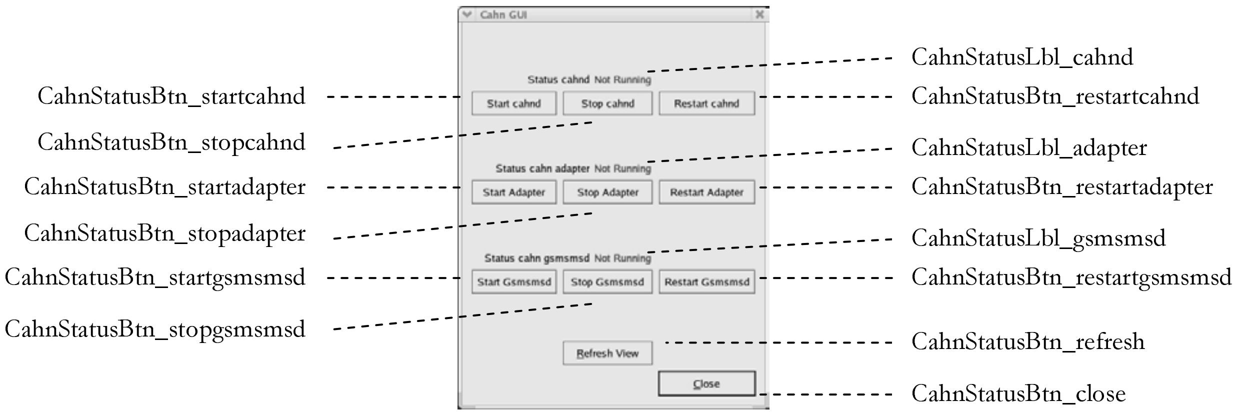


Figure 6.9: Cahn_status screenshot

This window serves to control the status of the involved daemons.

```
virtual void cahnStatusRefresh()
```

is called when the CahnStatusBtn_refresh is pressed and actualizes the labels CahnStatusLbl_cahnd, CahnStatusLbl_adapter and CahnStatusLbl_gsmsmsd, which show the current state of the appropriate daemons. This function is also called after each start, stop or restart process.

```
virtual void cahnStatusStartCahnd()
virtual void cahnStatusStopCahnd()
virtual void cahnStatusRestartCahnd()
```

are called, when CahnStatusBtn_startcahnd / CahnStatusBtn_stopcahnd / CahnStatusBtn_restartcahnd is pressed and starts/stops/restarts the cahnd daemon. The functions

```
virtual void cahnStatusStartAdapter()
virtual void cahnStatusStopAdapter ()
virtual void cahnStatusRestartAdapter ()
virtual void cahnStatusStartGsmsmsd()
virtual void cahnStatusStopGsmsmsd ()
virtual void cahnStatusRestartGsmsmsd ()
```

perform the same actions for the adapter and the gsmsmsd. When the CahnStatusBtn_close is pressed, accept() is called and the window is closed.

The Cahn_services class

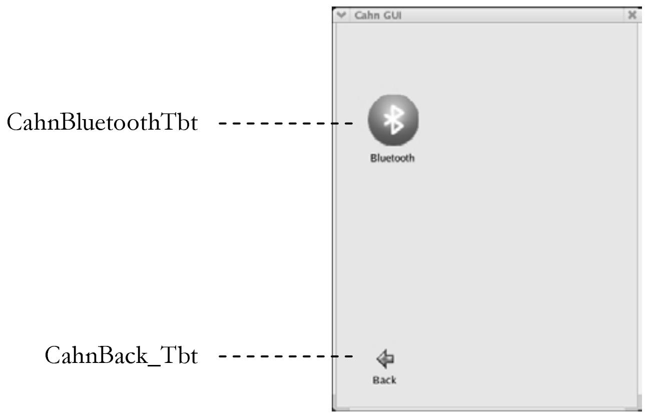


Figure 6.10: Cahn_services screenshot

This window is intended the offer the user the choice of different service classes. In this application only Bluetooth services are available, and therefore CahnBluetoothTbt is the only choice the user can make. When this button is pressed, the function

```
virtual void cahnBluetooth()
```

is called, which creates a new instance of the Cahn_bluetooth class and opens a new window. When the CahnBack_Tbt is pressed, accept() is executed and the window is closed. This brings back the main window of the CAHN GUI (class Cahn_gui).

The Cahn_bluetooth class

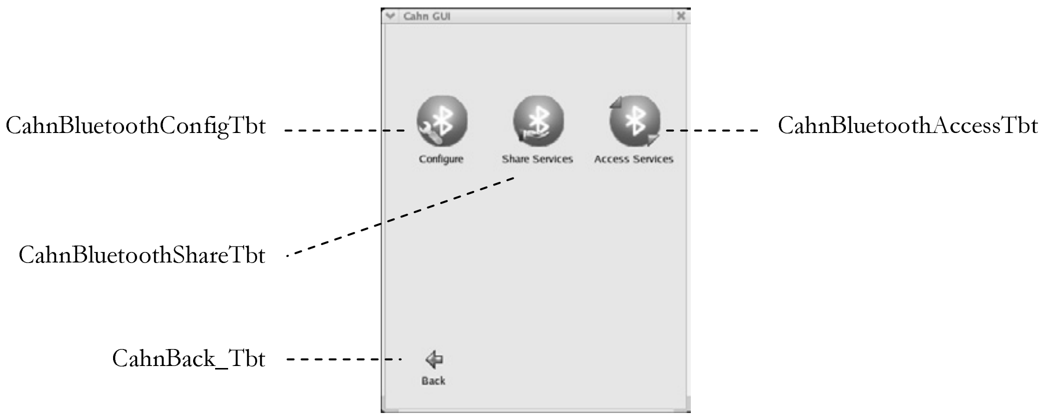


Figure 6.11 Cahn_bluetooth screenshot

The window showed by class Cahn_bluetooth provides the control of the CAHN Bluetooth services. The user can configure the Bluetooth related settings, share services and access to shared services in its proximity. The method

```
virtual void cahnBluetoothConfig()
```

is called, when the CahnBluetoothConfigTbt button is pressed. It creates a new instance of the Cahn_bluetooth_config class and brings up a window for Bluetooth related configuration settings.

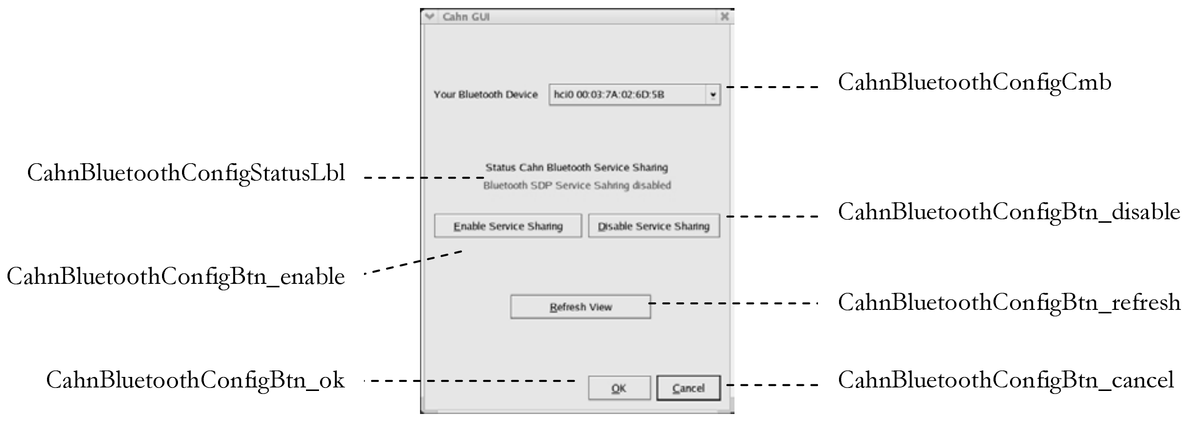
```
virtual void cahnBluetoothShare()
```

is called when the CahnBluetoothShareTbt button is pressed. This function creates a new instance of the Cahn_bluetooth_share class, which provides a window to share Bluetooth services.

```
virtual void cahnBluetoothShare()
```

is the responsive handler function upon a CahnBluetoothAccessTbt pressed event and creates a new instance of CahnBluetoothAccess. The button CahnBack_tbt again calls accept() which closes the window and brings up the previous window, which is provided by the class Cahn_services.

The Cahn_bluetooth_config class



Your Bluetooth Device: hci0 00:03:7A:02:6D:5B

Status Cahn Bluetooth Service Sharing
Bluetooth SDP Service Sahring disabled

Cahn GUI
Enable Service Sharing
Disable Service Sharing
Refresh View
OK
Cancel
CahnBluetoothConfigCmb
CahnBluetoothConfigStatusLbl
CahnBluetoothConfigBtn_disable
CahnBluetoothConfigBtn_enable
CahnBluetoothConfigBtn_refresh
CahnBluetoothConfigBtn_ok
CahnBluetoothConfigBtn_cancel

Figure 6.12: Cahn_bluetooth_config screenshot

The Cahn_bluetooth_config class is used to control the Bluetooth related parts of CAHN. In this window the Bluetooth device for CAHN can be chosen (this is useful, when more than one Bluetooth adapter is present) and the status of service sharing can be controlled.

```
virtual void cahnBluetoothConfigEnable()
virtual void cahnBluetoothConfigDisable()
```

are the functions called, when service sharing is enabled or disabled by clicking the respective button. Enabling service sharing will start a Bluetooth SDP server process and will register the CAHN service to this server. Disabling will deregister the CAHN service from the Bluetooth SDP server.

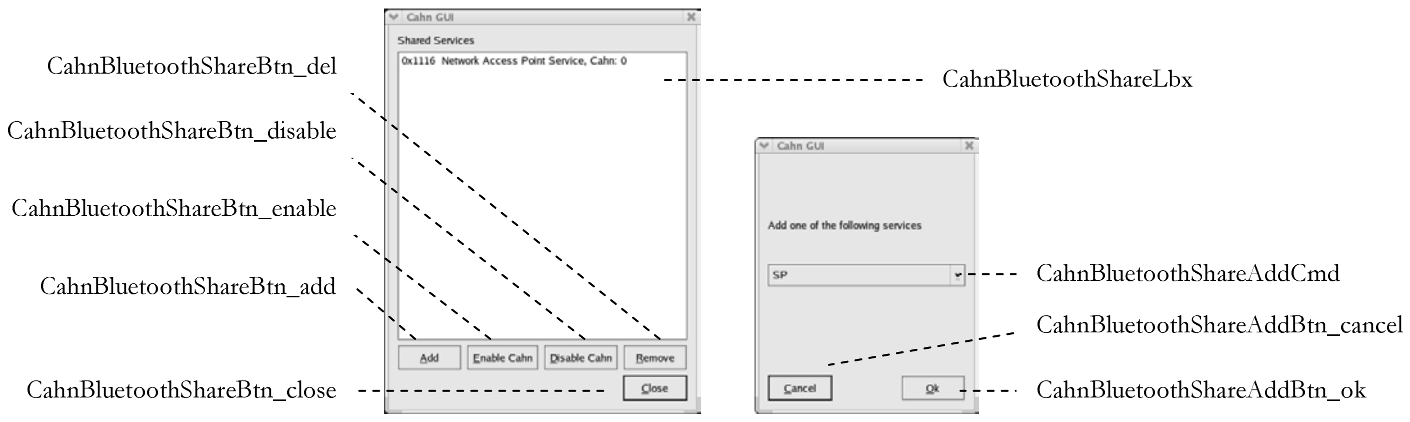
```
virtual void cahnBluetoothConfigRefresh()
```

is used to refresh the CahnBluetoothConfigStatusLbl, showing whether service sharing is enabled or not. This function is also called, when one of the service sharing control buttons is pressed.

```
virtual void cahnBluetoothConfigOk()
```

is invoked by pressing the ok button in the window and will store the Bluetooth device address in a file. This address is used by the CAHN application. Then accept() is executed to close the window. The Cancel button will only call accept() without saving the Bluetooth device address. Note, if the state of the service sharing has been changed, it will not be updated, when cancel is pressed!

The Cahn_bluetooth_share and Cahn_bluetooth_share_add classes


CahnBluetoothShareBtn_del
CahnBluetoothShareBtn_disable
CahnBluetoothShareBtn_enable
CahnBluetoothShareBtn_add
CahnBluetoothShareBtn_close
Cahn GUI
Shared Services
Add
Enable Cahn
Disable Cahn
Remove
Close
CahnBluetoothShareLbx
Cahn GUI
Add one of the following services
SP
Cancel
Ok
CahnBluetoothShareAddCmd
CahnBluetoothShareAddBtn_cancel
CahnBluetoothShareAddBtn_ok

0x1116 Network Access Point Service, Cahn: 0

Figure 6.13 Cahn_bluetooth_share and Cahn_bluetooth_service_share_add screenshots

The classes Cahn_bluetooth_share and Cahn_bluetooth_share_add and the windows they provide offer an interface to the local Bluetooth SDP server. The list box in the window of Cahn_bluetooth_share contains the services that are currently registered at the Bluetooth SDP server. In addition to the Service UUID and the human readable representation, the items in the list box also show whether the service is enabled for CAHN.

The button CahnBluetoothShareBtn_del can be used to disable service sharing. Therefore the function

```
virtual void cahnBluetoothShareDel()
```

is executed, which deregisters a service on the Bluetooth SDP server. To change the CAHN status of a service

```
virtual void cahnBluetoothShareEnable()
virtual void cahnBluetoothShareDisable()
```

are used, when the appropriate button is pressed. These functions do nothing else than add or remove the CAHN Bluetooth service attribute to (from) the chosen Service Record on the local Bluetooth SDP server. To add a new service to the Bluetooth SDP server, the button add is pressed, which in turn executes

```
virtual void cahnBluetoothShareAdd()
```

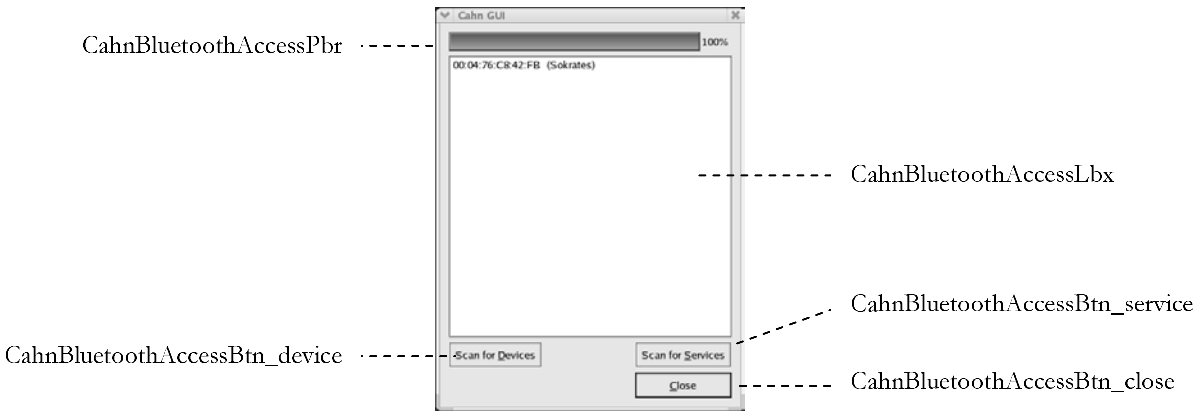
This function creates an instance of the Cahn_bluetooth_share_add class, whose window can be seen on the left of figure 6.13. In this window a service can be chosen with help of the combo box CahnBluetoothShareAddCmb. If the ok button is pressed,

```
virtual void cahnBluetoothShareAddOk()
```

is run, and the service will be added. When cancel is pressed, the window is closed, without adding the service to the local Bluetooth SDP Server. In order the share services the service sharing must be enabled before in the Cahn_bluetooth_config window.

The close button in the window of Cahn_bluetooth_share closes the window and returns to the window of the Cahn_bluetooth class.

The Cahn_bluetooth_access class



00:04:76:C8:42:FB (Sokrates)

Figure 6.14: Cahn_bluetooth_access screenshot

This class provides a window for the user to easily access the devices. The user can scan for devices and services on them. The progress bar, CahnBluetoothAccessPbr shows the progress of the device search. When the user intends to search for devices

```
virtual void cahnBluetoothAccessDevice()
```

is run. This function performs an Inquiry and prints the devices in Bluetooth proximity to the list box. Additionally the function does request the names of the devices, which answered to the Inquiry with their Bluetooth address. To search for services, provided by a certain devices, the user can click on the button “Scan for Services” which invokes

```
virtual void cahnBluetoothAccessService()
```

This function does create an instance of the Cahn_bluetooth_services class, which comes up with a new window. To return to the previous window, the close button can be used.

The Cahn_bluetooth_access class

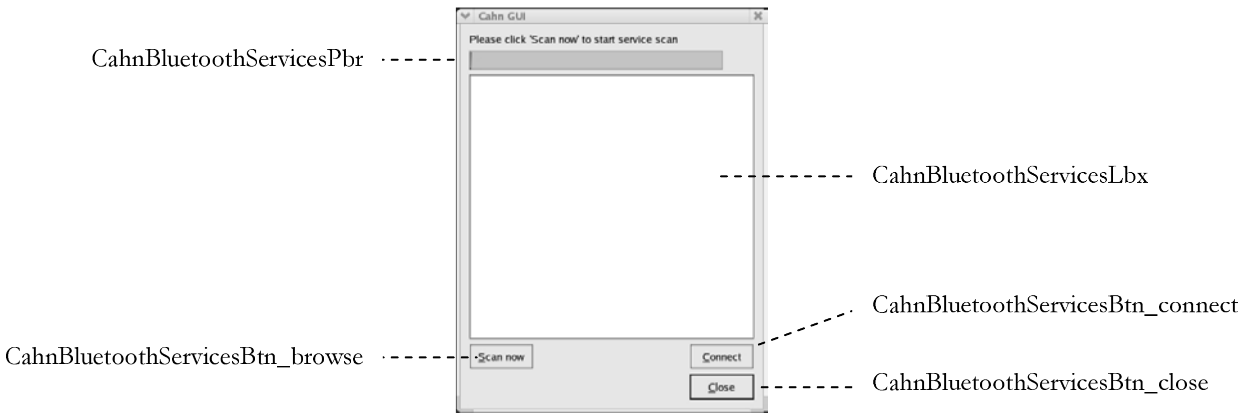

Cahn GUI
Please click 'Scan now' to start service scan
CahnBluetoothServicesPbr
CahnBluetoothServicesLbx
CahnBluetoothServicesBtn_connect
CahnBluetoothServicesBtn_browse
Scan now
Connect
Close
CahnBluetoothServicesBtn_close

Figure 6.15: Cahn_bluetooth_services screenshot

Within this window the user can scan a chosen device for services. To do so, he can press the button “Scan now”, which calls

```
virtual void cahnBluetoothServicesSearch()
```

which issues a service browse request to the remote Bluetooth SDP server. The services found are added to the list box with their CAHN status indicated by the presence or the absence of the CAHN Bluetooth service attribute in the Service Record. The progress bar visualizes the status of the current service browse request. If CAHN enabled services are found the user can press the connect button, what will start

```
virtual void cahnBluetoothServicesConnect()
```

This function can now, with help of CAHN, establish the connection for the user. To achieve this, a CAHN Bluetooth Service Request message is generated, a connection to the cahnd daemon established and the request sent to the cahnd, which will handle the request and give back the CAHN Bluetooth Service Response or the CAHN Error message, in case of an error. When the GUI receives a positive response, the other party accepted the request and sent back the response including the PIN. This PIN has been extracted by the local CAHN adapter and added to the PIN database. As both parties now have a common PIN a Bluetooth connection to the chosen service can be established. This connection establishment is also handled by this function, and the GUI will show up with a “You are connected” message or with an error message, indicating what went wrong. The exact procedure to establish a connection to a certain service will depend on the implementation of the respective Bluetooth profile. A “LAN Access using PPP” connection is established in a different way than a NAP connection and so on. Therefore this function must provide different mechanisms to handle the different services. Up to now only one service is implemented, which is the NAP.

7 EVALUATION

In this chapter measurements made with the application running on a test-bed are presented. The out coming of the tests supplies important knowledge about the practical behaviour of the provided application and further it will help to evaluate the chosen design. After the description of the test-bed, the connection establishment times of the application will be presented. In a next step these measurements are analyzed and explained, always using an empirical approach.

7.1 THE TEST BED

On the server side just an ordinary desktop machine was chosen. No special server hardware or architecture is used at all. This desktop computer is equipped with an AMD Athlon processor running at 1 GHz. The system has 512 MB of RAM. Further the system provides a 10/100 MBits Ethernet adapter and USB 1 support. The chosen Bluetooth interface hardware for the establishment of the PAN connection is a 3Com USB dongle and another 3Com Bluetooth USB dongle is used for communication with the mobile phone, which is an Ericsson T68i.

On the client side a Toshiba Portégé P4010 Laptop has been made available for this diploma work. This Laptop has an Intel processor running at 733 MHz and is equipped with 128 MB of RAM. The Laptop provides a built-in Bluetooth adapter for the establishment of the PAN connection and a TDK Bluetooth USB dongle for the interaction with the mobile phone, which is also a T68i.

Both computers are running RedHat Linux 9.0, which ships with the Bluez device drivers and kernel space portions of the Bluez Bluetooth stack. Further the Bluez SDP and the Bluez PAN Profile implementations are installed on both machines. The server is setup with an Internet connection and has the needed bridging mechanisms configured, which are necessary to provide the PAN Profile Network Access Point (NAP) scenario.

The mobile phones are attached via the respective Bluetooth dongles using the RFCOMM profile provided by Bluez. This profile emulates a RS-232 conform serial connection, which builds the basis to exchange AT commands the gsmsmsd program uses to send and receive SMS messages.

Additionally, both computers are running a window manager based on the XFree86 X window system. This is necessary for the GUI provided with the application.

7.2 CONNECTION ESTABLISHMENT TIMES

In a first step, the time is measured the implemented application takes for a Bluetooth connection setup with CAHN integrated. The client has already detected the server with help of an Inquiry and it knows about the services the server does provide. The service which was used is the access point service, offered by the PAN profile. Time measurement starts, when the connection setup is invoked on the client by pressing the connect button in the window implemented by the Cahn_bluetooth_access class. On the server a ping to the client's PAN IP address is invoked and as soon the first ping request is replied, the connection is established and the time measurement stops. 20 measurements were made and figure 7.1 shows the results of these tests.

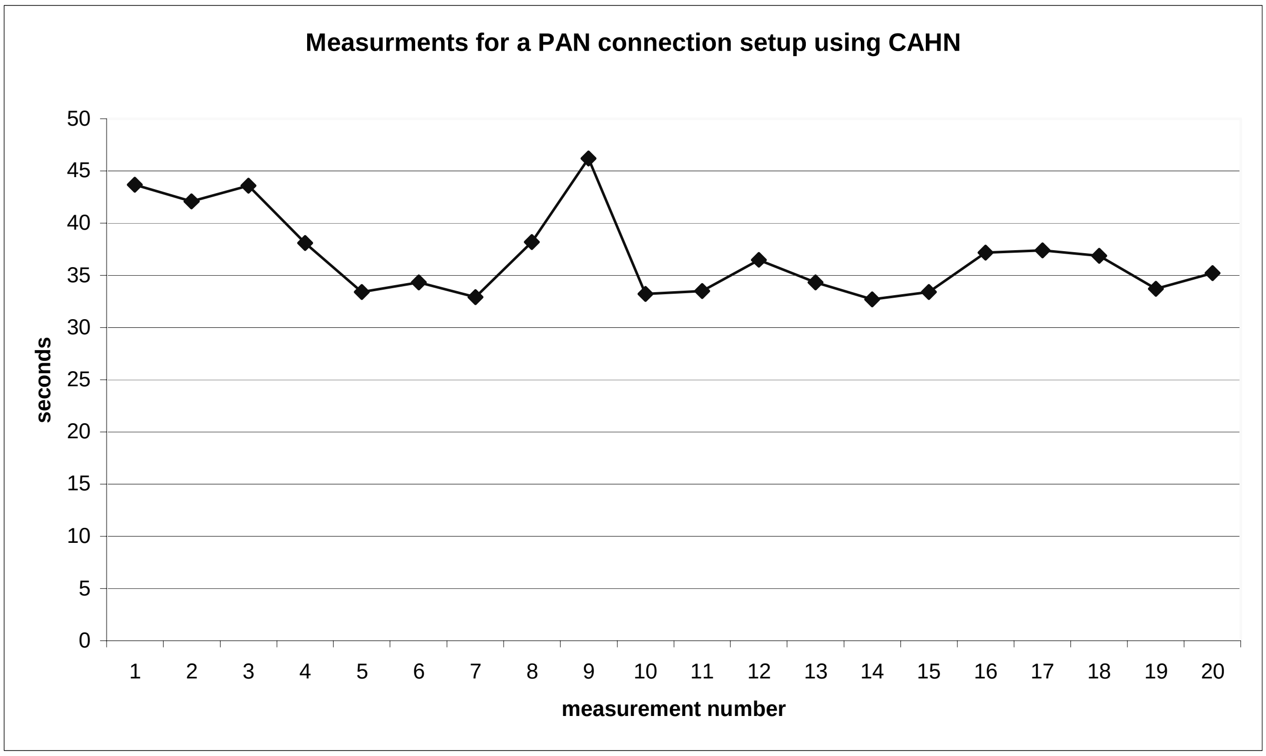


Figure 7.1 PAN connection establishment times

In the figure can be seen that the times measured for a PAN connection setup range from 32.7 seconds in minimum up to 46.2 in maximum. These values are only the measured values, and they do not indicate the best respectively the worst case. Best and worst case estimation do not make sense, as can be seen later in the section about the SMS message exchange delays. The average connection establishment time is 36.9 seconds. The values presented are quite distributed and a high variance is observed.

In the following sections, it is tried to detect what precisely influences the connection establishment times. For that purpose the whole setup procedure is divided in single actions which are treated further to understand what makes the establishment time and what integrates the high insecurity factor in the connection establishment times.

The whole procedure starts with the GUI creating a CAHN Bluetooth connection request and sending it to the CCM. The CCM then detects, that the messages has to be relayed to a remote host and hands the message over to the CAHN adapter. The CAHN adapter converts the message into an SMS and stores the SMS to a spool, where the gsmsmsd checks periodically for spooled SMS to be sent over the GSM network using the attached mobile phone. These are the actions on the client side. The extraction of the client's system log file presented in figure 7.2 shows that the actions between connection establishment invocation and the writing of the SMS to the spool directory are indicated to happen in the same second. These actions are therefore not influencing the total connection establishment times and are ignored in the further analysis.

```
Mar 23 10:01:02 MobileNode1 cahnd[4580]: Got a CAHN Bluetooth
Service Request
Mar 23 10:01:02 MobileNode1 cahnd[4580]: Got a remote request for
+41793776746
Mar 23 10:01:02 MobileNode1 cahn_adapter: Response Socket for
cahnd: 0
Mar 23 10:01:02 MobileNode1 cahn_adapter: Got a remote packet for
+41793776746
Mar 23 10:01:02 MobileNode1 cahn_adapter: This is the SMS:
cahn%0%2%68%4374%00:03:7A:02:6D:5B%00:04:76:E1:A9:BF
Mar 23 10:01:08 MobileNode1 gsmsmsd[4610]: sent SMS to
+41793776746 from file /root/sms_spool//message
```

Figure 7.2 Extraction of the CAHN client's system log file

But the time passed from writing the SMS to the spool and actually sending it is interesting. Figure 9.2 shows 6 seconds for that period, which can be interpreted as the time taken by spooling mechanism of gsmsmsd. The manual of the gsmsmsd tells that the program checks the spool every 5 seconds. It will become clear, why the time in the log file is bigger than these 5 seconds. But the spooling mechanism has to be taken into account on both sides, the client and the server (for the response) and the average is 2.5 seconds on every side, which means about 5 seconds for spooling in total.

Once the SMS is sent, we have to take the SMS message exchange delays into account. More on this follows in the next chapter. When the SMS arrives on the server's mobile phone, the gsmsmsd invokes the CAHN SMS adapter, which converts the SMS into a CAHN message, which is handed to the CAHN adapter. The CAHN adapter relays the message to the CCM, which will hand it to the CAHN connector to handle it. This relaying of the message does not influence the total connection establishment times and is ignored. The CAHN connector does perform several actions to handle the connection request. First it checks the database for a possible existing registration of the request. Then a random, unique PIN is generated. Next it is checked whether the device is in range of the Bluetooth radio. What follows is the update of the PIN database and finally a response is generated and returned to the CCM.

The check for an existing registration, the check whether a generated PIN is unique and the update of the PIN database are performed with help of SQL queries to a MySQL database. These MySQL queries are known to perform very fast and are ignorable for the total connection establishment time as well as random PIN generations. What in contrast matters is the check whether the client is in the range of the Bluetooth radio. This is done with help of the Bluetooth Inquiring mechanism. To increase the probability to really hit a client in the vicinity the Inquiry is repeated for a certain time. It is a trade-off between speed and quality of the Inquiry result. Within the implementation an Inquiring period of 5 seconds has been chosen. These 5 seconds are relevant for the total connection establishment times.

After completion of these actions, the CCM has received a response from the CAHN connector, which is relayed to the CAHN adapter and sent as an SMS back to the client. On the client side, the gsmsmsd gets the SMS and invokes the CAHN SMS adapter, which converts the SMS to a CAHN response and sends it to the CAHN adapter, which in turn relays the message to the CCM. The CCM will then invoke the CAHN connector to handle the response. To do so, the CAHN Connector updates the PIN database with the PIN included in the CAHN response. Then the response is handed via the CCM to the requesting GUI, for indication to the user and as the GUI acts on behalf of the user, to invoke the Bluetooth PAN connection establishment.

Now the four factors, which mainly influence the total connection establishment time, are identified:

1. Spooling mechanism (can be interpreted as a constant value of 5 seconds for both sides)

2. The Inquiry performed on the server (constant value of 5 seconds)

3. SMS message exchange delays (measurements are necessary and will follow)

4. Bluetooth PAN connection setup (measurements are necessary and will follow)

7.3 SMS MESSAGE EXCHANGE DELAYS

Before we come to the results of the measurements of the SMS message exchange delays, it has to be stated that in the worst case the SMS can be queued for one week in the operator's network. This one week is the maximum possible validity duration that can be set for an SMS. Further SMS as service is not guaranteed, so that it could be possible that SMS messages get lost, even if such a loss has never been observed within the application development and testing phase. For that reasons it is really hard, to estimate the average SMS message exchange delay. To cope with this issue, 20 consecutive measurements were made to gain an idea about the average message exchange delay. For that purpose, the time was measured from the moment the send button was pushed on the sending mobile phone until the sound notification about an incoming message occurred on the receiving mobile phone. Both mobile phones used for this test are Sony Ericsson T68i. While doing these measurements, it could be observed, that the size of the message has an impact on the message exchange delay. Figure 7.3 illustrates the outcome of the tests for message sizes of 0, 57 respectively 160 characters.

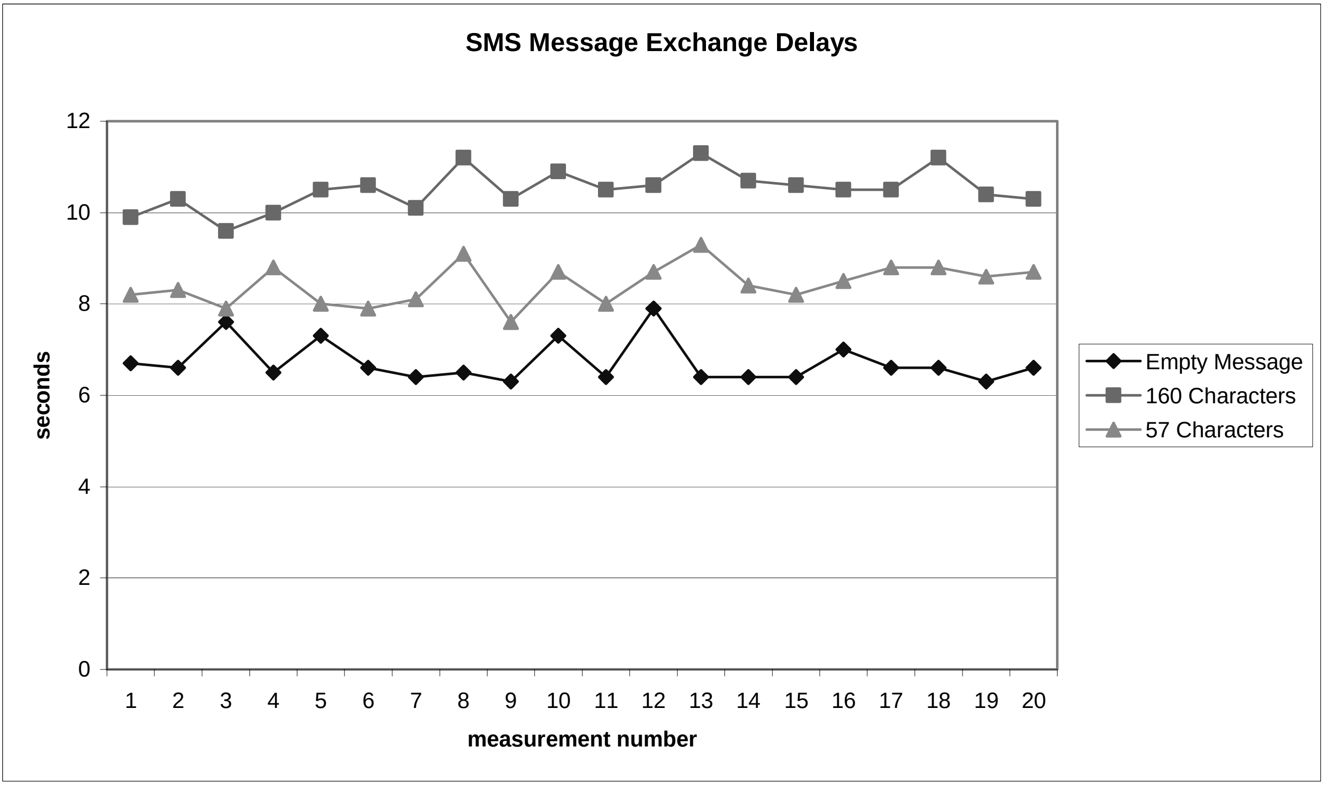


Figure 7.3 SMS message exchange delays

The size of 57 characters has been chosen as it represents the average length of a CAHN SMS.

From the values presented in the diagram it can be clearly observed, that the size of the message has an impact on the average message exchange delay. The average message exchange delays for the different messages are:

1. Empty message: 6.7 seconds

2. Message containing 57 characters: 8.4 seconds

3. Message containing 160 characters: 10.5 seconds

This result is quite surprising, as the amount of the additional data is rather small. Further analysis showed that the time for inserting an SMS into the GSM network is changing, depending on the size of the SMS. This was the reason to assume that the SMS insertion time could be hardware dependant. Reference measurements with a Nokia 7650 mobile phone confirmed that assumption, as the average message exchange delay for an SMS with 160 characters sent with this mobile phone was 14.4 seconds.

This hardware dependant behaviour for SMS sending is also the reason, why the log file excerpt, presented in figure 7.2, indicates an SMS sending time of 6 seconds, even if the spooling mechanism consumes at maximum 5 seconds. The additional second is caused by the time it takes for the mobile phone to upload the message to the GSM network. As soon as the message is uploaded, the mobile phone indicates this to the gsmsmsd, which generates the log file entry.

Nevertheless we focus on the measurements taken with the Sony Ericsson mobile phones, as these phones were taken for the measurements of the total connection establishment times. So the average message exchange delay of an SMS is 8.4 seconds.

7.4 BLUETOOTH PAN CONNECTION ESTABLISHMENT TIMES

As the fourth critical factor for the total connection establishment time, the setup duration for a PAN connection has been identified. Again it is hard to estimate an upper bound for the time such a connection establishment takes, as within the tests no timeouts occurred. The source code of the Bluez PAN implementation has not been further analyzed, so that no definitive statement about the existence of timeouts can be made. For the estimation of the average PAN connection establishment time the same approach as for the SMS message exchange delay and the total connection establishment delay estimations were used. 20 measurements were made from the moment the client invokes a connection establishment until the moment the first ping from the server to the client is responded. Figure 7.4 illustrates the results of the tests. Now it can be seen, that the high variance in the total connection setup times is highly dependent on the high variance of the values measured for the PAN connection setup, even more as the SMS message exchange delays do not show such a high variance and the other critical parameters are treated as constant values. Further it could be observed, that the setup of a PAN connection takes less time, if the link is not encrypted. These values are not interesting for this diploma work, as it clearly focuses on the establishment of a secured connection. The average time measured for the establishment of a secured PAN connection is 6.5 seconds.

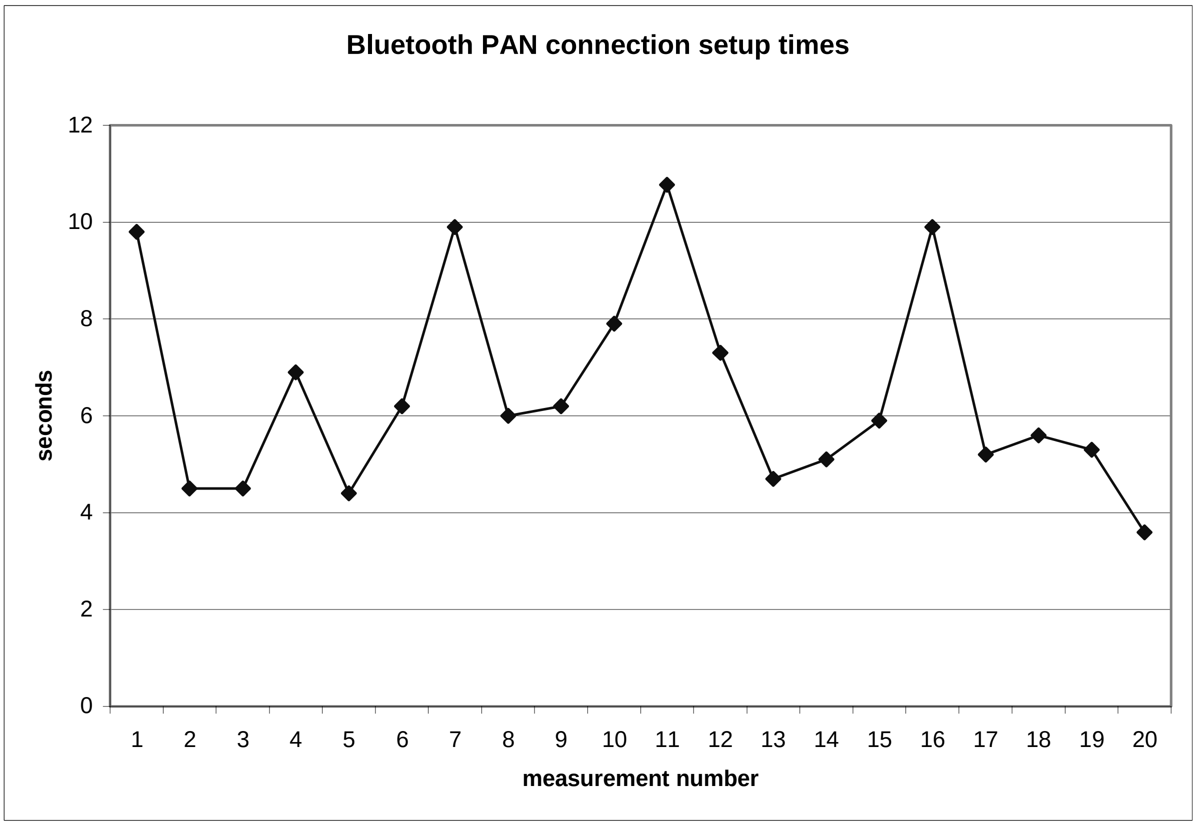


Figure 7.4 Bluetooth PAN connection setup times

7.5 PERFORMANCE SUMMARY

To draw conclusions, first the average values of the four critical factors are added up and compared to the average total connection setup time:

• Spooling mechanism:	5 seconds
• SMS message exchange delay (CAHN request):	8.4 seconds
• Server Inquiry:	5 seconds
• SMS message exchange delay (CAHN reply):	8.4 seconds
• PAN connection establishment:	6.5 seconds
Total:	33.3 seconds

So the sum of the estimated average values is 33.3 seconds. Compared with the average measured value of 36.9 seconds for the total connection setup, the rest of the implementation consumes about 3.6 seconds. This is the sum of all the other parameters influencing the duration for a connection setup. Among these parameters also the ping round trip time has to be mentioned, which indicated in the tests sometimes values about 2 seconds. So it can be stated that the application does not add too much overhead to the connection establishment. To validate this thesis some reference measurements were made, where the CAHN server returned an error message to the client, and thus the PAN connection was not established. This was done by measuring the time from the connection establishment invocation until the indication of the error message in the GUI. The reference values showed an average time of 29.5 seconds with a small variance. Compared with the above sum of the average values minus the average time of a PAN connection, an overhead of 2.7 seconds results for the application in case of an error. This further allows drawing the conclusion, that the real overhead is even smaller and the estimated overhead is also a result of the measurements inaccuracy.

Nevertheless the total connection establishment time is not satisfying at all and improvements have to be done in order to make the application useful. The best starting point for future improvements would be the replacement of SMS messages for the transport of CAHN messages. If we have a look at the critical factors, the spooling mechanism, which is also a consequence of the use of SMS, and the SMS message exchange delays make about 22 seconds of the whole connection establishment, which means about 60%. Further the Inquiry test of the server could be removed, even if this would result in registrations of devices, which are not in range. This would be another gain of 5 seconds, or 13%. In contrast to those easily changeable factors, the PAN connection establishment is a factor, which is not simple to influence and still would add a lot of overhead to the application.

8 CONCLUSIONS

8.1 LESSONS LEARNED

Overall, the work on this diploma work showed the handling problem in today's networking technologies, which are caused by the big spectrum of different access technologies and the trend towards mobile devices. The formation of networks is no longer static and this dynamism brings in a strong configuration and management issue.

CAHN is a very promising approach to ease the configuration and management of dynamic networks and can offer a basic authentication, which provides a big accessibility. The reuse of broadband links for the data transmission enables the combination of a high bandwidth and low coverage data link with a low bandwidth and high coverage signalling link.

The dynamism of mobile networking has also an impact on the traditional server client role, which is not applicable on those topologies. The use of service discovery protocols enables the deployment of dynamic client and server applications for these needs and brings in an easy service sharing and service accessing mechanism. As mentioned the service discovery protocols have a lack of security at the present stage.

The marriage of the CAHN approach and service discovery protocol driven environments can cope with this security problems, as CAHN provides the necessary authentication mechanism, which is also accessible in spontaneously established networks, with no additional link to the Internet and therefore no access to many traditional authentication methods. On the other hand, CAHN profits from the integrated service discovery protocol, as CAHN enabled nodes can be discovered and their capabilities can be indicated.

These advantages are underlined by the implemented demonstrator, which integrates a basic CAHN implementation and the Bluetooth SDP. Further the demonstrator serves as a first proof of the concept to integrate CAHN in service discovery protocols.

This integration would offer the means to ease the configuration and establishment of spontaneous networks. In addition to the setup related issues, the integration of CAHN and service discovery protocols can also provide an authentication mechanism as a base for secure service deployment and access of services. The dynamic client server mechanisms will conclude the integration and enable participating users to share and easily access services in the network.

8.2 FUTURE WORK

The demonstrator application showed the applicability of CAHN in a Bluetooth SDP environment. As the Bluetooth SDP is a very basic approach of service discovery, no final conclusions on the value and the behaviour of the integration of CAHN into more sophisticated service discovery protocols can be drawn. Further investigations in this direction will therefore be necessary to examine whether the approach scales for networks envisioned by other service discovery approaches.

It is probable that an integration of CAHN in Jini is not or not easily possible in existing Jini applications, as they are not open source. Salutation on the other hand can be quite hard as the specifications of Salutation are voluminous and existing implementations rare. The use of UPnP for the service sharing and access platform is not suggested, as UPnP clearly targets devices and not services. Last but nor least it would be a good choice to take SLP as the basis for a future implementation, as SLP offers a very clear and straight forward definition of service discovery as well as an Open Source implementation already provided by modern GNU/Linux distributions.

With help of the knowledge gained in this diploma work and the help of further investigations with the OpenSLP implementation of SLP, it could maybe be possible to adopt the approach for Salutation driven environments.

Additionally the multicast issue was not occurring in the demonstrator, so that no conclusions can be drawn from this application about this important issue of service discovery protocols. CAHN could be used as an alternate communication channel to multicast channels, which could increase the scalability of service discovery protocols. Implementations based on the former suggested OpenSLP could help to analyze the applicability of CAHN to cope with the multicast problem.

It is therefore obvious that other service discovery protocols have to be analyzed for a possible CAHN integration. But not only in the domain of service discovery protocols have to be done further investigations, but also in the CAHN implementation itself.

Using SMS for the transport of CAHN messages over the cellular network is not a satisfying solution, as seen in the evaluation of the demonstrator applications. To provide better performing applications other mechanisms than SMS must be applied. USSD [27] promises to be a better approach for this task. It also has native support for broadcasting, which could be used for service discovery protocols multicast messages.

Furthermore, the demonstrator application does not provide implementations for the security procedures between the operator and the CAHN enabled cellular device. These security mechanisms have to be evaluated with help of the operator, which can provide access to certain resources, helping to authenticate the user. Fact is, that the existing trust relation between the user and the operator can serve as the basis for further trust relations, but the mechanisms, how this is achieved are not clear, yet.

The role of the operator in the global scenario still has to be defined and thus also the degree of interaction of the operator. It is imaginable, that the operator can serve as lookup service for service discovery protocols, can provide key management facilities or can serve as a proxy to access further authentication mechanisms.

To conclude this diploma work it can be stated, that the provided demonstrator application shows to be promising approach. But no definitive statements about its applicability in practice can be drawn. More experience in the topic has to be collected, and still a lot of work, which has be identified above, has to be done to prove the concept.

9 REFERENCES

[1] C. Perkins, „IP Mobility Support for IPv4“, August 2002, RFC 3344

[2] Microsoft Corporation, “Virtual Private Networking: An overview”, September 2001, white paper

[3] M. Danzeisen, “Secured Mobile IP Communication”, May 2001, Diploma work at the University of Bern

[4] Bluetooth SIG, “Specification of the Bluetooth System”, Version 1.2, November 2003

[5] ANSI/IEEE standard 802.11, “Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications”, 1999

[6] ETSI standard, “European digital cellular telecommunications system (Phase 2)”, March 1999

[7] Sun Microsystems, “Jini Architecture Specification”, December 2001

[8] Sun Microsystems, “Java RMI – Distributed Computing for Java”, white paper

[9] R. Srinivasan, “RPC: Remote Procedure Call Protocol Specifications Version 2”, August 1995, RFC 1831

[10]Microsoft Corporation, “Understanding Universal Plug and Play: A White Paper”, June 2000, white paper

[11]J. Cohen, S. Aggarwal, Y. Goland, “General Event Notification Architecture Base: Client to Arbiter”, September 2000, Internet Draft

[12]W3C, “Soap Version 1.2”, June 2003, Recommendation

[13]W3C, “Extensible Markup Language (XML) 1.0 (Second Edition)”, October 2000, Recommendation

[14]Y. Goland, T. Cai, P. Leach, Y. Gu and S. Albright, “Simple Service Discovery Protocol”, October 1999, Draft

[15] C. Lee and S. Helal, "Protocols for Service Discovery in Dynamic and Mobile Networks", International Journal of Computer Research, Volume 11, Number 1, pp. 1-12 ISSN 1535-6698, 2002

[16] C. Bettstetter and C. Renner, "A Comparision of Service Discovery Protocols and Implementation of the Service Location Protocol", Institute of Communication Networks of the University of Munich, http://wwwtgs.cs.utwente.nl/Docs/eunice/summerschool/papers/paper5-1.pdf

[17] B. Pascoe, "Salutation-Lite: Find-and-Bind Technologies for Mobile Devices", Salutation Consortium, June 1999, white paper

[18] Bluetooth SIG, "Generic Access Profile", November 2003

[19] Bluetooth SIG, "Service Discovery Application Profile", November 2003

[20] Bluetooth SIG, "Bluetooth Assigned Numbers – Service Discovery Protocol", November 2003

[21] B. Miller, "Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer", July 1999, Bluetooth Consortium 1.C.118/1.0

[22] IETF SrvLoc, working group

[23] E. Guttman, C. Perkins, J. Veizades, M. Day, "Service Location Protocol, Version 2", June 1999, RFC 2608

[24] A. Campbell, "The Progress of Dynamic Resource Discovery: Resource Discovery Protocol vs. Service Location Protocol", April 2000

[25] M. Danzeisen, R. Rodellar, T. Braun, S. Winiker, „Heterogeneous networking establishment assisted by cellular operators", The Fifth IFIP TC6 international conference on mobile wireless communications (MWCN 2003), Singapore, October 27-29 2003

[26] Bluez website, http://www.bluez.org

[27] GSM Recommendation 02.90, "Unstructured Supplementary Services Data – USSD"

[28] G. Richard, „Service and Device Discovery Protocols and Programming", ISBN 0-07-137959-2

[29]E. Maghsoudi, “CAHN Protokollnachrichten über das GSM Netz“, August 2003, Informatikprojekt Universität Bern

[30]S. Winiker, „Service Discovery Protocols“, October 2002, Technical Report, Swisscom Innovations AG