Master Thesis

Application-Layer Communication Protocol for Home Automation

Dinko Hadzic

June 15, 2004

University of Oslo
Faculty of Mathematics and Natural Sciences
Department of Informatics

Abstract

Chipcon [CHI], a Norwegian company that designs, produces and markets high performance and cost-effective radio frequency integrated circuits (RF-ICs), wanted to develop a new communication protocol for monitoring and control scenarios that would be applicable with all their products.

The thesis proposes a new application-layer communication protocol for home automation named the Device Control Protocol (DCP). Being a completely transmission-layer independent and simple but flexible and extensible communication protocol that allows cost-effective embedded implementation in a wide range of application areas, DCP meets and exceeds the requirements given by Chipcon. In order to simplify DCP-based application implementations, the thesis also defines a standardized application program interface (API) for the protocol, which hides the complex details of the underlying transmission layer and provides a uniform interface to upper layers regardless of the selected transmission technology.

As an open, universal home automation protocol, the Device Control Protocol (DCP) provides a solid foundation for further development and industrialization by other other manufacturers seeking a simple but flexible communication solution.

The thesis also explores and confirms the possibility of using a mobile phone as a short-range remote control in home automation. Due to the platform independency and a strong market acceptance, Java Platform 2 Micro Edition (J2ME) is the recommended software platform for hosting the remote control applications. The recommended wireless technology is Bluetooth.

Various prototype systems are developed to illustrate the results of the thesis and demonstrate the practical application of the Device Control Protocol (DCP).

"Every new beginning comes from some other beginning's end"

Conficius, 550-478 BC

Preface

This Master Thesis is written at the University of Oslo, Faculty of Mathematics and Natural Sciences, Department of Informatics [UIO] and the University Graduate Center at Kjeller [UNI]. The project is carried through during one semester (30 credit points), it started January 15th and concluded June 15th, 2004.

The project is defined by a commercial company, Chipcon AS [CHI], that has contributed with hardware equipment, technical expertise and consultations.

Parts of the thesis are also published on the official thesis website, available online at http://folk.uio.no/dinkoh.

I would like to thank my supervisors Knut Øvsthus, Øyvind Janbu and Pål Spilling for valuable help, guidance and counselling throughout the project period. I would also like to thank Hans Klouman for creating the electrical power relay chip put to use in one of the prototype systems.

Oslo, June 15, 2004

Dinko Hadzic

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

> "You come home from work and as you approach the house a retinal analyzer recognizes you, opens the door and greets you by turning on the lights. In the kitchen, you walk towards your fridge, which informs you that you are missing a few ingredients from the recipe for tonight's dinner."

The term home automation covers processes, systems and technologies that make the home more comfortable, convenient, safe and efficient, as illustrated in the quotation above [SHA]. Home automation covers a wide range of applications like home lighting, security systems and access control, home theatre and entertainment control, and ranges from simple scenarios like lighting control to complex, integrated systems.

This thesis is defined by Chipcon [CHI], a Norwegian company that designs, produces and markets high performance and cost-effective radio frequency integrated circuits (RF-ICs) for use in a variety of wireless applications. Motivated by the potential reduction in installation cost and complexity as no new wiring is needed and the possibility for battery powered operation, the use of wireless home automation systems is expected to increase significantly during the next few years [Sol]. In this context, Chipcon wanted to evaluate the applicability of three short-range wireless technologies having the potential of gaining ground within home automation: IEEE 802.15.4, ZigBee and Bluetooth. In addition, a short theoretical review of

several other common home automation technologies that exist on the market, both wired and wireless, should be presented.

Offering a complete product family of RF-ICs with varying characteristics, Chipcon wanted to develop a new application-layer communication protocol for home automation monitoring and control scenarios, applicable with all their RF-ICs. If possible, the protocol should be completely independent of the underlying transmission layer, thus supporting both wireless and wired transmission technologies. The protocol should as simple as possible in order to enable cost effective embedded implementations. At the same time, the protocol should be flexible enough to support a diversity of application areas and products. Chipcon wanted a considerable part of the project to be invested into designing and programming of one or more prototype systems that demonstrate the characteristics and application possibilities of the proposed communication protocol.

The thesis also explores the possibility of using a mobile phone as a short-range remote control in home automation. Mobile phones are constantly increasing the complexity and processing power, and more and more phones incorporate one or more short-range wireless technologies. The mobile phone should be able to communicate with the home automation appliances through the protocol specified in this thesis. The proposed solution should be compatible with mobile phones from different manufacturers. If possible and allowed by the time frame, a simple remote control application should be implemented and tested on a mobile phone.

1.2 Thesis Definition and Scope

The main goals of this thesis are as follows:

- Present a short theoretical review of several home automation technologies that exist on the market, both wired and wireless.
- Present an in-depth introduction to three short-range wireless communication technologies having the potential of gaining ground within home automation, IEEE 802.15.4, ZigBee and Bluetooth, and evaluate and discuss their applicability in home automation scenarios.
- Specify an application layer communication protocol for suitable for use in home automation. The protocol should be simple enough to

allow low cost implementations but flexible enough to allow a broad range of products. If possible, the protocol should be independent of the underlying transmission technology.

- Explore the possibility of using a mobile phone as a short-range wireless remote control in home automation.

- Demonstrate the results of the thesis by implementing one or more prototype applications.

1.3 Related Work

Extensible Automation Protocol (XAP) [XAP] is an open protocol intended to support the integration home automation devices. The initial implementation focuses on IP based networks, although [XAP] claims to support other network types. Note that the solution proposed in this thesis has not been influenced by XAP in any ways, the proposed solution is developed completely from scratch to suit the requirements given by the thesis definition.

The author is unaware of any other attempts to define an open, universal application-layer protocols for home automation. Although some alternatives for industry automation exist, they are considered beyond the scope of this thesis focusing on home automation, which has other protocol requirements than industry automation.

A minor part of the thesis also explores the mobile phone usage as a short-range remote control in home automation. The paper [HK03] discusses the home appliance control from a mobile phone, focusing on one software platform (Java Platform 2 Micro Edition (J2ME) [J2M]), and one wireless technology (Bluetooth [SIG01b]). This thesis focuses on presenting and evaluating several alternatives, both software platforms and wireless technologies, and recommending a solution based on the evaluation. The paper [KT02] discusses the use of Bluetooth technology in wireless home automation networks, and how such networks can be controller from a WAP (Wireless Application Protocol) browser on a mobile phone. In contrast to [KT02] which focuses on "long-range" remote control from potentially many kilometers away from the home appliance itself, this thesis focuses on short-range remote controol limiting the distance between phone and home appliance to typically 10-100 meters.

1.4 Report Overview

Chapter 2, Home Automation Systems The chapter gives a short overall review of several home automation technologies that exist on the market, both wired and wireless.

Chapter 3, Short-Range Wireless Technologies The chapter gives an in-depth introduction to three short-range wireless technologies, IEEE 802.15, ZigBee and Bluetooth, and briefly evaluates the applicability of each technology in wireless home automation scenarios.

Chapter 4, Device Control Protocol (DCP) This chapter defines a new application-layer communication protocol for home automation called Device Control Protocol (DCP), and proposes a generic application program interface (API) for the protocol. DCP is an open and simple but flexible and extensible communication protocol independent of the underlying transmission technology.

Chapter 5, Mobile Phone in Home Automation The chapter explores and discusses the possibility of using a mobile phone as a short-range remote control in home automation. The chapter presents and evaluates a number of common software platforms attempting to find platforms applicable for creating remote control applications that are compatible with phones from different manufacturers. The chapter also discusses the applicability of short-range wireless technologies available on mobile phones today: Bluetooth, IrDA Infrared and IEEE 802.11 WLAN.

Chapter 6, Prototype Systems The chapter presents the purpose, technical architecture and user instructions of prototype applications implemented during the thesis. The prototype systems offer several reference implementations of the Device Control Protocol (DCP) and the proposed application program interface (API) based on various transmission technologies. The source code of all prototype systems can be found on the accompanying CD-ROM.

Chapter 7, Discussion The chapter discusses the main results of the thesis, and identifies the further work to be done.

Chapter 8, Conclusion The chapter presents the main conclusions of the thesis.

Appendix A This appendix lists the contents of the accompanying CD-ROM.

Chapter 2

Home Automation Systems

The chapter presents a short review of several wired and wireless home automation technologies that exist on the market today.

2.1 Introduction

The home automation systems can be classified into two categories according to the transmission medium they use, wired or wireless.

Wired communication either exploits the existing electrical wiring in the home or requires a separate cabling between devices. Devices using the power lines are inexpensive and easy to install, but they are vulnerable to electrical noise on power lines generated by other devices. Devices requiring a separate cabling typically offer larger bandwidth and more reliable communication, but they are more expensive and the installation procedure is more complex.

Wireless communication is based on either infrared (IR) or radio frequency (RF) signals. IR communication requires line-of-sight communication, while RF offers omnidirectional communication where signals penetrate walls and other obstacles. The products have the potential of being battery powered, thus being very flexible in terms of mobility. The installation procedure is also relatively simple as no preexisting cabling infrastructure is required.

2.2 X-10

X-10 is a proprietary communication protocol that allows devices to talk to each other using the existing electrical wiring in the home. X-10 is developed by the company with the same name [X10a].The products are inexpensive and easy to install by simply plugging the device into the electrical outlet. X-10 has existed for over 20 years, and many X-10 compatible products are available today. X-10 is able to address up to 256 unique devices. However, if there is too much electrical noise on the power lines generated by some other electrical device, the X-10 devices might have problems communicating. A number of devices are known to interfere with X-10 devices [X10b], some of these are:

- Televisions
- Computers
- Game console machines
- Motors: refrigerator, heating systems, pumps etc.
- Cell phone chargers, toothbrush chargers etc.

Assessment X-10 is inexpensive, widely deployed and easy to install, but it also has some drawbacks. In certain contexts, the X-10 communication is unreliable because the technology is affected by the operation of other electrical appliances nearby, and X-10 signals can get lost. X-10 products are typically installed into electric outlets, which restricts the potential installation locations considerably, making it unsuitable for certain home automation scenarios. X-10 is a proprietary technology, although the company also sells the X-10 products through other companies, under other brand names [Met01].

2.3 LonWorks

LonWorks [INT] [Tie00] is a networking platform for control systems in building, industry, utility and home automation, introduced by Echelon [ECH] in 1988. Today, LonWorks is a de facto standard in commercial building automation and industry control. LonWorks has existed for more than 15 years,

and it has become widely deployed. More than 1500 companies are developing LonWorks products. LonWorks devices communicate using the LonTalk protocol. Based on the Open Systems Interconnect (OSI) protocol stack reference model, LonTalk defines 7 protocol layers. The protocol provides a set of services that allow the application program in a device to send and receive messages from other devices over the network without needing to know the topology of the network or the names, addresses, or functions of other devices. Networks can range in size from 2 to 32000 devices. LonTalk can be implemented upon many medium types including power lines, twisted pair, radio frequency (RF), infrared (IR), coaxial cable and fiber optics, although the most common choice is twisted pair cable.

Assessment Through the years, LonWorks has proven to be a reliable technology. However, the home control systems are typically smaller and simpler networks where important properties are low installation complexity, low battery consumption and low product price. LonWorks was designed with none of these in mind as main issues. LonWorks was design to be reliable, and to cover a diversity of potential products and markets. It seems like the home automation is a secondary market for LonWorks. In order to fit into home automation, products need to be physically small, simple and inexpensive. They must be easily installed (plug-and-play). LonWorks products are relatively large (physical size) and quite complex to install and configure. The price of a typical LonWorks product is relatively high [LWP]. LonWorks products must be installed by trained and approved LonWorks system integrators, which increases the total system complexity and cost. Although LonWorks theoretically might use a variety of physical mediums, the common choice is twisted pair cable. Wiring a house is expensive. Even if some wireless LonWorks products exist, they are relatively expensive and not optimized for battery powered operation because of the characteristics of the LonTalk protocol.

2.4 Konnex

The Konnex Association [KNX] was established in 1999 by joining together three organizations:

- BatiBUS Club International (BCI) [BCI]

- European Installation Bus Association (EIBA) [EIB]
- European Home Systems Association (EHSA) [EHS].

The goal of the organization is promote a single standard for home and building automation, called KNX. Today, the Konnex Association has approximately 100 member companies. The KNX standard is based on the EIB standard, and supports two transmission medium types, twisted pair cable and power line. The next version of the standard (version 1.1) will also include support for RF and IR media. The protocol stack is based on the OSI model and specifies the link layer, the network layer, the transport layer and the application layer.

Assessment Products based on the KNX protocol from the Konnex Association are not widely deployed in home automation. The KNX technology is mostly used in commercial building automation. However, the Konnex Association has plans to expand into the residential market. The Konnex Association has relatively strong industry support, specially in Europe [KNX].

2.5 Z-Wave

Z-Wave is developed by a commercial company, Zensys [ZEN]. Z-Wave is a proprietary wireless RF-based communications technology designed for control and status reading applications. Z-Wave offers duplex, reliable communication in a mesh network topology and operates at the rate of 9.6 Kbit/s. All Z-Wave communication happens on a single RF channel with a predefined frequency. Z-Wave implements a proprietary routing protocol allowing the devices to forward data packets from one device to another towards the correct destination.

Assessment Although the bandwidth of the Z-Wave is relatively low, it should be sufficient to cover the majority of home automation scenarios. Z-Wave allows implementations of battery powered devices and supports wireless routing, making it well suited for home automation. Z-Wave is a closed, proprietary technology. The deployment of the technology is limited. A potential disadvantage in environments with a lot of interference is that Z-Wave has no mechanisms to change the communication frequency in order to

find a frequency minimizing the amount of noise, making Z-Wave less robust against interference.

2.6 Consumer Electronics Bus (CEBus)

Consumer Electronics Bus (CEBus) [CEB] is an open set of communication protocols for home networks. CEBus is developed by the CEBus Industry Council (CIC) and standardized by the Electronics Industry Association (EIA). The CIC is a non-profit organization established in 1994 by Honeywell, Intel, Microsoft and Thomson Consumer Electronics. The CEBus standard supports power line communication (PLC), twisted pair (TP) cable, coax cable, RF and Infrared transmission media.

Assessment Few CEBus products exist on the marked, although the CEBus technology is more than 10 years old. In addition, the existing CEBus products are relatively expensive.

Chapter 3

Short-Range Wireless Technologies

This chapter provides an in-depth presentation of three short-range wireless technologies, IEEE 802.15.4 [LRW03], ZigBee [ZIG] and Bluetooth [SIG01b], and discusses their applicability in home automation scenarios. The desirable characteristics of a wireless technology suitable for use in home automation are:

- Low power consumption
- Low complexity
- Reliable transmission
- Secure transmission
- Cost effective implementation

3.1 IEEE 802.15.4

This section provides an in-depth presentation of the IEEE 802.15.4 [LRW03] technology and a short evaluation in the context of home automation.

3.1.1 Overview

The IEEE 802.15.4 [LRW03] standard specifies the physical (PHY) and media access control (MAC) layer for simple, low-cost radio communication networks, offering low data rates and low energy consumption. The purpose of the IEEE 802.15.4 specification is to provide a standard for ultra-low complexity, ultra-low cost, ultra-low power consumption, and low data rate wireless connectivity among inexpensive devices, as stated in [LRW03].

IEEE 802.15.4 provides a reliable communication protocol, and defines both a star and a peer-to-peer network topology. The standard uses carrier sense multiple access with collision avoidance (CSMA-CA) to avoid packet collisions. Two device types are possible, a full-function device (FFD) and a reduced-function device (RFD). A FFD device is capable of being the network coordinator implementing the complete protocol stack, while a RFD is a simpler device with a minimal protocol stack implementation.

The transmission distance is expected to range from 10 to 100 meters, depending on output power and the surrounding environment. The transmission can be optionally encrypted using Advanced Encryption Standard (AES). In order to increase battery life, the standard allows some devices to deactivate both the transmitter and the receiver for over 99% of their operating time [Cal04].

3.1.2 Physical Layer (PHY)

The main responsibility of the physical layer (PHY) is to control the radio transceiver. The layer also measures the energy level within the current channel, and provides the link quality indication (LQI) for received packets. Before sending packets on air, PHY optionally performs a CSMA-CA to identify if the channel is busy. PHY is responsible for transmitting and receiving packets on correct channel.

In order to provide flexibility for a range of applications, IEEE 802.15.4 operates in three frequency bands at different rates, offering flexibility to a range of applications. The frequency band properties are summarized in Table 3.1. IEEE 802.15.4 specifies a total of 27 communication channels across the three frequency bands. Not that the 868/902 MHz band requires a compliant device to be capable of operating in both frequency bands. This choice has been taken in order to minimize the number of potentially incompatible products on the market. The 868/902 MHz bands are likely to be less

Frequency band (MHz)	Bit rate (Kbit/s)	Number of channels	Geographical region
868 – 868.6	20	1	Europe
902 – 928	40	10	North America, Australia
2400 – 2483.5	250	16	Worldwide

Table 3.1: Frequency bands and data rates

crowded and offer better quality of service (QoS), but they are not available worldwide. The 2.4 GHz band is available worldwide.

IEEE 802.15.4 devices use direct sequence spread spectrum (DSSS) technique to increase the bandwidth of a transmitted signal, resulting in improved communication reliability.

3.1.3 Medium Access Control (MAC) layer

The Media Access Control (MAC) layer has several responsibilities. The layer is responsible for generating and synchronizing to the optional network beacons. The layer provides an association and disassociation mechanism, and provides a reliable link between two devices. It also offers optional MAC layer security and maintains a GTS (Guaranteed time slot) mechanism for devices that require a constant rate and fixed delays.

The MAC layer supports creation of two types of network topologies:

- Star topology
- Peer-to-peer topology

The network topologies are illustrated in Figure 3.1, taken from [LRW03]. In the star topology, all communication is controlled by the network coordinator. Any full-function device (FFD) can create its own network by becoming a Personal Area Network (PAN) coordinator, as specified in [LRW03]. Peer-to-peer topology allows more complex communication scenarios. Any FFD device might communicate with any other FFD device. It is possible to implement routing protocols in this topology. Reduced-function devices (RFD) might also participate in the network, but only as peripheral devices. They can not relay packets and participate in the routing mechanisms. Peer-to-peer networks are beyond the scope of the 802.15.4 standard.

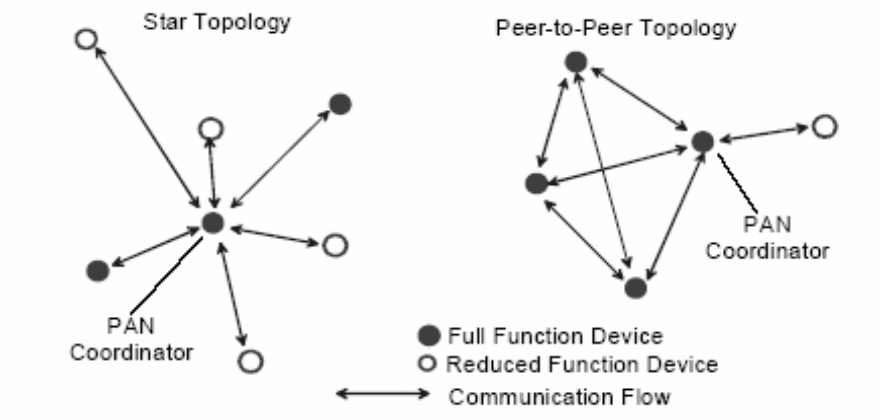

Figure 3.1: IEEE 802.15.4 network topologies: star and peer-to-peer

3.1.4 Home Automation Evaluation

The IEEE 802.15.4 technology is designed and optimized specifically for home and building automation and similar applications, and it is therefore well suited for building wireless networks. With the final specification being released in 2003, IEEE 802.15.4 is still a relatively new technology. Not many physical implementations of the standard or products based on the technology have been released yet, however the technology seems to be gaining ground continuously, strengthened by the ZigBee Alliance initiative to define higher communication protocols based on the IEEE 802.15.4 standard. The technology fills the need for a standardized, globally available low cost and low power short-range wireless technology that provides reliable and secure communication. It operates in three frequency bands at three different transmission rates, offering the manufacturers flexibility to make the optimal choice for their application.

3.2 ZigBee

This section provides an in-depth presentation of the ZigBee [ZIG] technology and a short evaluation in the context of home automation.

3.2.1 Overview

ZigBee [ZIG] defines the network, application and security layers suitable for use in building automation, industrial, medical and residential monitoring and control applications for wireless networks based on the IEEE 802.15.4 technology. Examples of ZigBee applications:

- Lighting control
- Automatic Meter Reading
- Wireless smoke detectors
- Heating control
- Home security
- Environmental controls
- Industrial and building automation

The ZigBee specification is still under development, and the first final version is scheduled to be released by the end of 2004. ZigBee is developed by the ZigBee Alliance [ZIG], a non-profit organization with membership open to all. The Alliance targets to define a global specification for reliable, cost-effective, low power wireless applications. The ZigBee alliance has today more than 70 members, and the number is continuously growing.

ZigBee devices are expected to have a low duty cycle and to be inactive most of their operating time. Combined with the low power consumption of the IEEE 802.15.4 technology, the users can expect the batteries to last for months and even years.

ZigBee defines three types of devices:

- ZigBee coordinator
- ZigBee end device
- ZigBee router

The ZigBee coordinator is responsible for setting up and maintaining the ZigBee network. It stores information about the network and the network participants. The ZigBee coordinator is typically not battery powered and it is listening continuously.

The ZigBee end device is designed and optimized for battery powered operation, and allows devices to be inactive in periods of time in order to minimize power consumption. The end device is able to search for available ZigBee networks and synchronize to one of these.

The Zigbee router device participates in the network by routing messages towards their correct destination.

ZigBee offers two network topologies:

- Star network topology
- Mesh network topology
- Tree network topology

The ZigBee network topologies are illustrated in Figure 3.2, which is created by the ZigBee Alliance [ZIG]. The star topology is defined by the IEEE 802.15.4 [LRW03] standard. In the star topology, all communication is controlled by the network coordinator, as explained in Section 3.1. The mesh and tree topologies make it possible to extend the communication by allowing multihop communication.

3.2.2 ZigBee Protocol Stack

The ZigBee protocol stack is illustrated in Figure 3.3. The lower layers of the stack are defined by the IEEE 802.15.4 standard, while the upper layers of the stack are defined by the ZigBee Alliance.

The size of the full ZigBee protocol stack implementation is expected to be less than 32 Kb for the ZigBee coordinator, and approximately 4 Kb for the ZigBee end device [Ada03]. The small size of the stack allows low-cost embedded systems implementations (i.e. implementations on inexpensive 8-bit microcontrollers).

Network Layer

The network layer is able to create a new ZigBee network and let other devices join or leave the network. The network layer of a ZigBee coordinator is

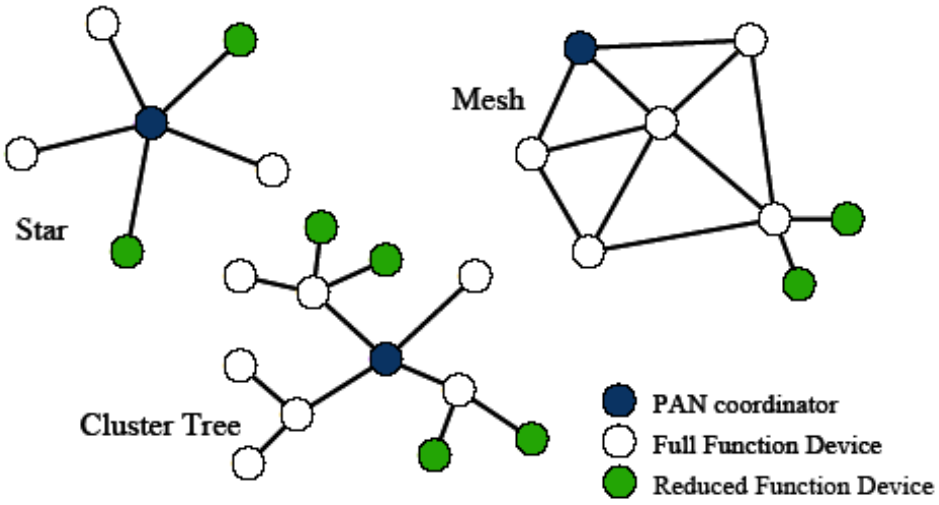

Figure 3.2: ZigBee network topologies: star, mesh and tree

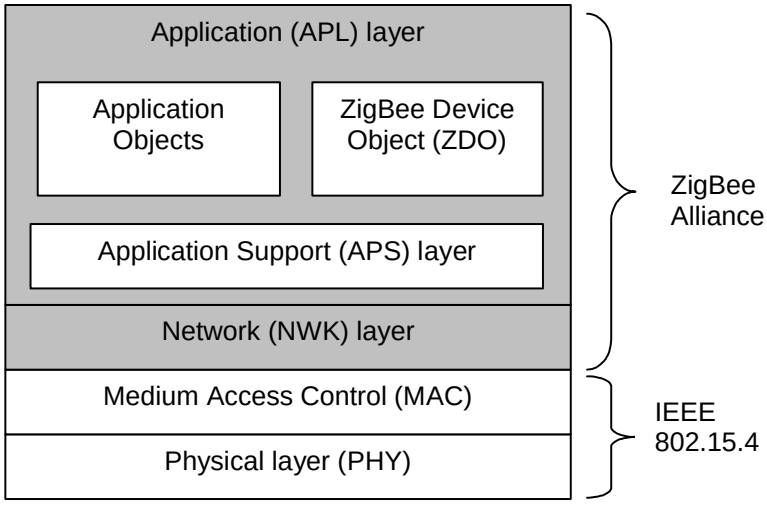

Figure 3.3: ZigBee protocol stack

responsible for assigning addresses to devices joining the network. The network layer implements the ZigBee routing algorithm, described by the ZigBee Alliance as "a hierarchical routing strategy with table driven optimizations where possible" [Ada03].

Application Layer

The application layer consists of the application support layer (APS), the ZigBee device object (ZDO) and the manufacturer-defined application objects.

The application support layer (APS) is responsible for maintaining tables for binding and forwards messages between bound devices. A binding is the ability to match two devices together based on their services and their needs. The layer is also responsible for device discovery, which is the procedure to discover other devices that are operating in the local area.

The ZigBee device object (ZDO) defines the role of the device in the ZigBee network (ZigBee coordinator or ZigBee end device). The ZigBee device object is also responsible for initiating and responding to binding requests.

The application objects are defined by the manufacturer, and they are defined by the manufacturer that implements the application. The ZigBee protocol stack supports up to 30 distinct application objects to be implemented at the same time.

ZigBee Security

ZigBee requires each layer in the stack to be responsible for its own security. However, this does not imply that the layer does the actual work. For example, applications can trust the network layer to secure their communications and the network layer can trust the MAC layer to secure its communications. Details about the ZigBee security mechanisms are not available yet, as the technology is still undergoing standardization.

3.2.3 ZigBee Profiles

ZigBee profiles provide target applications with the interoperability and intercompatibility required to allow similar products from different manufacturers to work seamlessly. ZigBee profiles are defined by the ZigBee Alliance [ZIG]

as an agreement on messages, message formats and processing actions that enable applications residing on separate devices to send commands, request data and process commands/requests to create an interoperable, distributed application. The first profile defined by the ZigBee is the home control lighting profile.

3.2.4 Home Automation Evaluation

ZigBee defining the network, application and security layers based wireless IEEE 802.15.4 networks. Fueled by the need for a standardized wireless system, ZigBee has gained relatively strong support by industry and is expected to become the de facto wireless standard for home automation. ZigBee specification is still undergoing specification, which means that the no ZigBee products exist yet, and that the final details about the technology are not available yet. ZigBee will support a wide range of products and application areas and guarantee interoperability between products. It will also support network routing and allow the network to be expanded incrementally.

3.3 Bluetooth

This section provides an in-depth presentation of the Bluetooth [SIG01b] technology and a short evaluation in the context of home automation.

3.3.1 Overview

Bluetooth [BTS] [SIG01b] is a short-range wireless technology intended to replace the cables between electronic devices, such as mobile phones, headsets and laptop computers. It was Ericsson Mobile Communications that started the development of the Bluetooth technology in 1994. In 1998 a group of companies formed the Bluetooth Special Interest Group (SIG) [BTS] that would work to define and promote the Bluetooth specification. Any company that wants to exploit Bluetooth commercially must become a member of the SIG organization. Version 1.0 of the Bluetooth specification was released in 1999. The IEEE organization adopted later parts of the Bluetooth stack into a formal IEEE 802.15.1 standard [LRW02] [WPAa].

Bluetooth offers omnidirectional wireless transmission of both voice and data in the globally available, license free 2.4 GHz Industrial, Scientific and

Medical (ISM) band. Devices are categorized into three different classes, according to their power consumption and transmission range:

- A class 3 device has 1 mW transmission power and a typical range of 0.1 – 10 meters
- A class 2 device has a transmission power of 1-2.5 mW and a typical range of 10 meters
- A class 1 device has a transmission power of up to 100 mW and a range of up to 100 meters

Bluetooth provides a bandwidth of 1 Mbit/s at the physical layer. It avoids interference and noise from other devices operating in the same frequency band by using the spread spectrum technique called frequency hopping. The communication changes the transmitting/receiving frequency 1600 times per second across 79 different frequencies.

3.3.2 Network Topology

Each Bluetooth connection has a master and a slave. By definition, the device that initiates the connection automatically becomes the master. The master can establish up to 7 simultaneous connections to other devices. A network of one master and up to 7 slaves is called a piconet. All devices on the same piconet follow the same frequency hopping and timing rules defined by the piconet master.

Two or more piconets can be linked together to create a scatternet. In a scatternet, one or more members participate in more than one piconet. However, they can only send and receive data in one piconet at a time. Such devices spend a few time slots on one piconet, and then few time slots on some other piconet etc. It is not possible for a single device to be master in two or more piconets. Although the scatternet topology makes it possible to multihop communication, the Bluetooth specification [SIG01b] does not specify any scatternet routing protocols.

Bluetooth piconet and scatternet topologies are illustrated in Figure 3.4.

3.3.3 Protocol Stack

The Bluetooth protocol stack is illustrated in Figure 3.5.

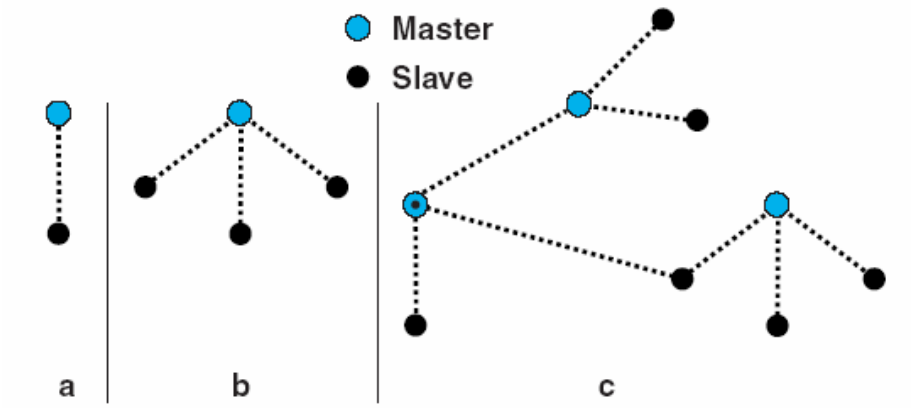

Figure 3.4: a) Point-to-point piconet b) Point-to-multipoint piconet c) Scatternet

The radio block is responsible for transmitting and receiving data packets on the physical channel by modulating and demodulating data on air.

Baseband is responsible for all access to the radio medium. The link controller (LC) is responsible for the encoding and decoding of Bluetooth packets from the data payload and parameters related to the physical channel, logical transport and logical link. The link controller carries out the Link Control Protocol (LCP) signaling.

The link manager is responsible for the creation, modification and releasing of logical links. The LM protocol allows the creation of new logical links and logical transports between devices when required, as well as the general control of link and transport attributes such as the enabling of encryption on the logical transport, the adapting of transmit power on the physical link, or the adjustment of QoS settings for a logical link.

Logical link control and adaptation (L2CAP) protocol provides a connection-oriented and connectionless data services to higher layer protocols, with segmentation, reassembly and group abstraction. It also functions as a multiplexer, by allowing multiple logical links on a single physical links, by implementing the concept of logical channels.

RFCOMM is the protocol for emulation of the RS232 serial port connections over L2CAP.

Wireless Application Protocol (WAP) allows devices to use data services of the underlying protocol stack, and access the Internet.

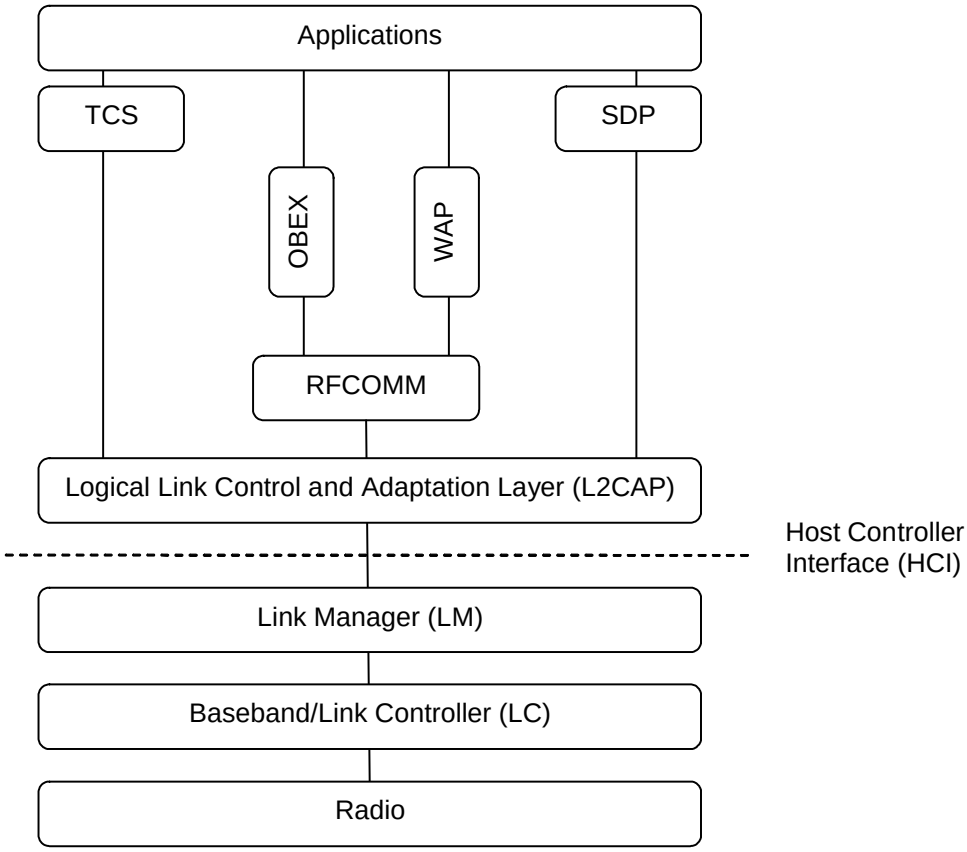

Figure 3.5: Bluetooth protocol stack

Object Exchange protocol (OBEX) is a session-layer protocol that supports exchange of objects (files) in a simple and spontaneous manner. It has client-server architecture. OBEX was developed by the Infrared Data Association (IrDA), and later adapted to Bluetooth.

Service Discovery Protocol (SDP) provides a way to search and discover what services the Bluetooth devices in the area nearby can offer.

Telephony Control Specification (TCS) provides telephony services.

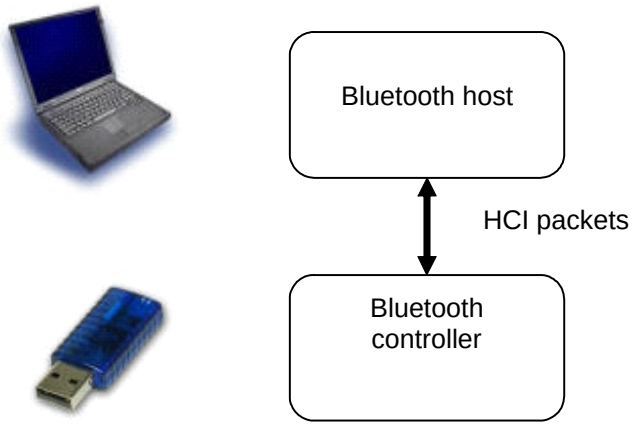

Figure 3.6: Host Controller Interface (HCI) architecture

3.3.4 Host Controller Interface (HCI)

A typical Bluetooth device consists of two parts: a host and a controller, communicating through a standardized host-controller interface (HCI). Although a common scenario, this separation is not mandatory, and devices might implement both host and controller protocol layers in a single unit, thus avoiding the need for communication through the HCI interface. The HCI architecture is illustrated in Figure 3.6. The host implements the upper layers of the Bluetooth protocol stack, while the Bluetooth controller implements the lower layers of the protocol stack. The HCI interface provides a uniform method of accessing the capabilities and configuration parameters of the lower layers of the stack.

3.3.5 Bluetooth Profiles

Bluetooth profiles describe how Bluetooth technology should be used in applications, and ensure interoperability between applications from different manufacturers. Profiles are arranged in a hierarchy where some profiles depend on others. Generic Access Profile (GAP) is root of the hierarchy, defining minimum basic functionality that all Bluetooth devices shall have. More information and a list of all profiles can be found in [SIG01a].

3.3.6 Home Automation Evaluation

Bluetooth is intended as a cable replacement technology between electronic devices, such as mobile phones, headsets and laptop computers, and is not optimized to suit the home automation needs. The Bluetooth connection and inquiry (device scan) procedure are time consuming. The maximum time for a connection procedure is 2.56 seconds, while the device scan takes up to 10.24 seconds, according to the Bluetooth Specification version 1.1 [SIG01b]. In many home automation scenarios, these delays are not acceptable. The Bluetooth SIG has the ambition to improve and shorten the connection and inquiry procedures, and the new version 1.2 [SIG03] of the Bluetooth standard shortens the inquiry and connecting procedure. Bluetooth allows up to 7 simultaneous connections, which might be insufficient in some scenarios. Bluetooth power consumption is too high for battery powered devices where battery is not expected to be charged periodically. Bluetooth makes it possible to create networks called scatternets where multihop communication is possible, but no routing protocols have been defined by the standard. At this time, there are no Bluetooth profiles targeting home automation, although some of the general Bluetooth profiles could be used instead.

Chapter 4

Device Control Protocol (DCP)

The main goal of the thesis was to develop a new application-layer communication protocol suitable for use in home automation monitoring and control scenarios, as defined by Chipcon [CHI]. This chapter presents the proposed solution, a new communication protocol called Device Control Protocol (DCP). The protocol has been designed, specified and implemented as a part of the thesis.

4.1 Overview

Device Control Protocol (DCP) is an application-layer communication protocol suitable for use in home automation, but also other similar areas like building and industry automation. For example, DCP can be used to transfer on/off commands from a switch to a lamp, or transfer temperature information between a thermostat and a heater unit. The protocol is developed to satisfy the requirements given by Chipcon [CHI] in the thesis definition.

The proposed protocol is independent of the underlying transmission technology and it is possible to use many kinds of transmission mediums like radio frequency (RF), twisted pair cable etc. The protocol is also independent of the underlying communication protocol. For example, DCP could be implemented upon TCP, Bluetooth RFCOMM etc. DCP requires the underlying protocol to provide reliable transmission. No transmission error detection or correction functionality is incorporated into DCP because it is an application layer protocol which relies on the underlying layer to guarantee correct transfer of DCP packets and indicate an error to the DCP layer if the transfer

could not be carried through.

The thesis proposes an application program interface (API) for DCP which is independent of the underlying transmission technology, simplifying the application implementation by providing a uniform interface and hiding the complex details of the underlying layer.

DCP supports a wide range of potential products and application areas. Besides standardizing the most common usage areas of the protocol, the protocol allows manufacturers to implement their own proprietary applications that are not covered by the DCP specification.

New transmission technologies and underlying communication protocols should be accepted by the DCP without problems, as DCP puts low requirements on the underlying technologies and communication protocols. New application areas can be covered by the manufacturers by exploiting the parts of the protocol reserved for proprietary usage, or by creating a new version of the protocol. DCP supports up to 256 different versions of the protocol. DCP communication includes a DCP version check in order to detect potential version inconsistencies.

Simplicity of the protocol was one of the key issues when DCP was designed. Because of the simplicity, DCP allows cost effective implementation on low cost embedded platforms.

4.2 Services, Ports and Bindings

The protocol introduces the concepts of services, ports and bindings, making it truly flexible and dynamic for use in a wide range of applications. The terminology is explained in following paragraphs.

Service A DCP service is an abstraction of a device's ability to perform some specific task. As illustrated in Figure 4.1, a device implementing a `SERVICE_TEMP_C` service has a capability to measure the temperature using the Celsius scale. A device implementing the `SERVICE_TIME` maintains the current time of day. Each device implements one or more services, making it possible to implement both very simple and more complex devices. There is no central service administration, and no devices know about all services of all devices in the neighborhood. Each device knows only about its own services, which are preinstalled when a device is manufactured. A device

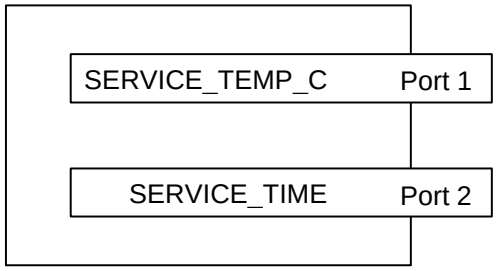

Figure 4.1: A DCP device implementing two services, SERVICE_TEMP_C and SERVICE_TIME at two different ports

can find out information about services offered by some remote device by performing a DCP service discovery procedure.

In order to ensure flexibility and support for a wide range of applications, DCP makes it possible to distinguish 65535 different services. Some services are standardized in the DCP specification (see Section 4.9), others are reserved for future versions of the DCP protocol. Manufacturers implementing the protocol are also free to define their own proprietary services in case these are not covered by this specification, at the expense of interoperability between devices from different manufacturers.

Port Each service is offered at a single port. A port is a simple integer functioning as a service multiplexer in scenarios where a device implements more than one service or multiple instances of the same service type (e.g. light control panel with multiple switches). The port concept makes possible implementations of more complex devices having several services. A port number uniquely identifies a service port within a device in cases where several services coexist on a device. A device that offers two services at two different ports is illustrated in Figure 4.1.

Binding The services of a device are offered to other devices through the binding concept. Bindings are logical connections between two services of same type. They are agreements between devices to notify each other if the value of a service is changed. A device can have one or more services, and therefore one or more bindings to a remote device, depending on its purpose and complexity.

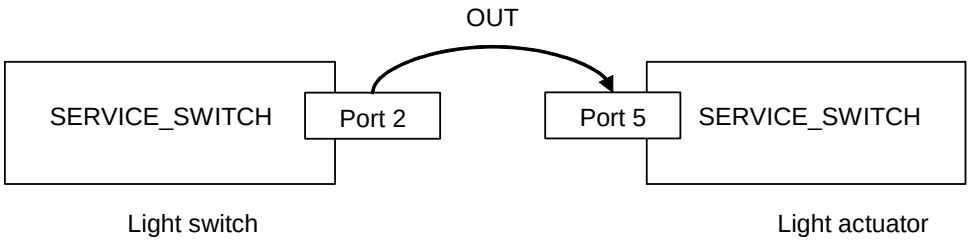

Figure 4.2: A DCP binding created by the light switch with the binding direction "out"

Each binding has a direction. The direction is either “in”, “out” or “in and out”. The direction is always considered from the viewpoint of the device initiating the binding procedure. If a device creates a binding with a direction “in”, then this specified service can be exploited by the remote device. If a device creates a binding with a direction “out”, then the device can make use of the service on the other device. The direction “in and out” lets the bound devices use each other’s services mutually. Figure 4.2 shows a scenario where a light switch creates a binding with the direction “out” to a light actuator. After the binding has been set up, the switch is ready to start controlling the actuator.

4.3 Addressing

DCP defines a simple address translation mechanism, introducing the 16-bit DCP addresses. The DCP address translation mechanism is illustrated in Figure 4.3. During the protocol development stage, it was important to keep the DCP as simple as possible, and at the same time it was desirable to offer a uniform API completely independent of the underlying transmission technology to the application. The DCP address translation mechanism can be seen as a compromise between these two requirements. The main objective of the DCP addresses is to provide a uniform addressing method to the application by being incorporated into the DCP API. The application only sees and operates with the 16-bit DCP addresses, regardless of the addressing method of the underlying transmission layer. The DCP layer is responsible for maintaining the address translation tables, which bind the DCP addresses to the

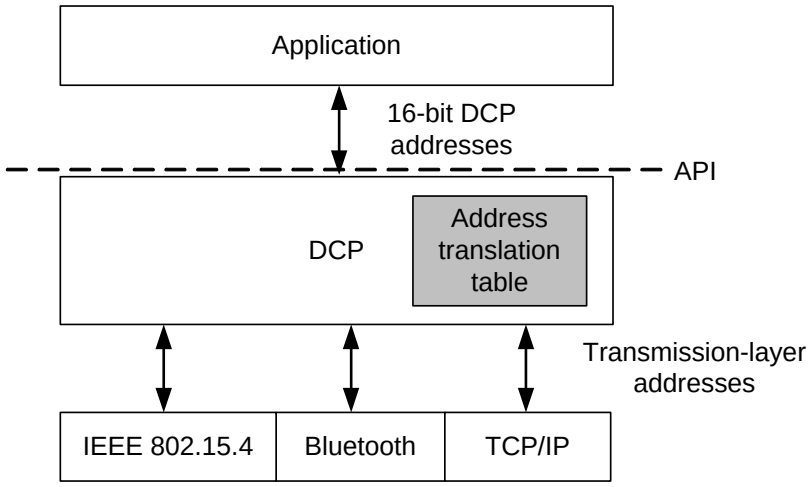

Figure 4.3: DCP address translation mechanism

corresponding unique device address defined by the underlying transmission layer, and use these for actual data transmissions.

The DCP addresses are never transmitted to any other device, but rather maintained internally by each device and used by the DCP API and the application implementation. This choice of not sending simplifies the address assignment by not needing to check whether the address is already used by some wireless other device, and allow each DCP implementation to decides how the DCP address translation mechanism is implemented, and which algorithm it will choose to produce DCP addresses.

4.4 Error Handling

DCP communication is based on request-response transaction. Each device that receives a request message shall answer with a corresponding response message. However, if the operation requested in the request message can not be carried out, the device shall respond with a corresponding error message instead of response message. Each error message carries an error reason.

The two types of DCP transactions are illustrated in Figure 4.4. DCP Client represents the device initiating the DCP transaction, while DCP server is the device that responds to the DCP request message.

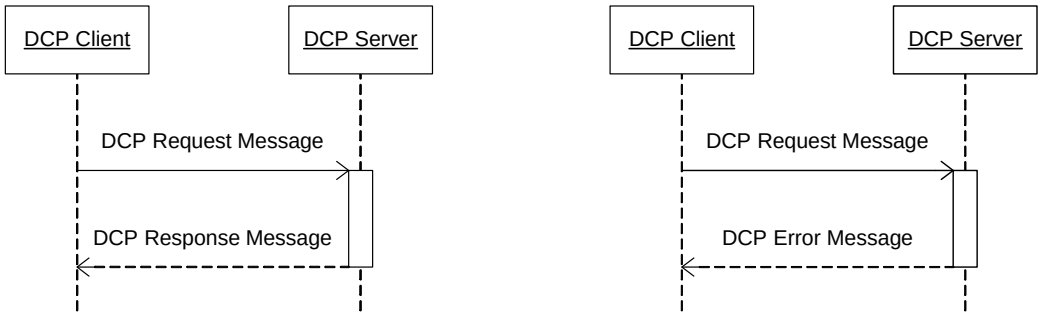

Figure 4.4: DCP request-response transactions: a successful scenario and an error scenario

The error handling mechanism described above concentrates on covering error situations at DCP layer. The mechanism does not cover situations where the error is generated by the transmission technology at the layer below DCP, which include:

- Connection error – it is not possible to create a connection to the specified device
- Transmission error – data could not be transmitted to the specified device
- Timeout error – requested operation has been canceled because of a timeout

One way of intercepting and handling these errors is through API function return values. The API functions would return special return values to indicate an operation failure to the application. Due to lack of time, the API function return values have not been standardized by this thesis.

An alternative to handling the transmission technology errors through protocol API function return values is to generate DCP error packets locally. For example, if the DCP layer tries to transmit a DCP message, and the operation fails, the DCP layer would generate a DCP error packet locally and deliver it to the higher layer. The error reason of the error message would indicate the failure of the transmission operation.

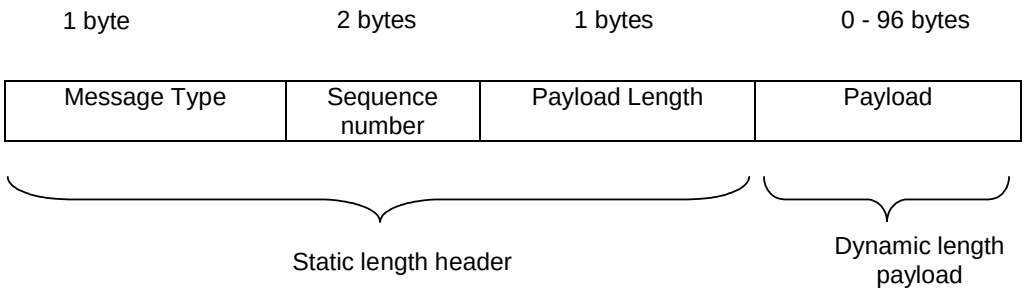

Figure 4.5: DCP packet format

4.5 Security

Communication security, including authentication, authorization, encryption and other security mechanisms are beyond the scope of this thesis.

4.6 Packet Format

DCP packet format is defined in Figure 4.5. The least significant byte is illustrated leftmost in the figure, and is transmitted first. The packet format is kept as simple as possible deliberately.

The Message Type field determines the purpose of the packet. DCP message types are defined in Section 4.8. The Sequence number field is used to ensure packet freshness, and is incremented by 1 for each request message sent. Response messages shall have the same sequence number as the corresponding request message. The Payload Length field defines the length of the Payload field. The Message Type field defines representation and formatting of the Payload field.

To ensure flexibility for potential future protocol changes and add-ons, DCP makes it possible to define up to 256 different message types, 65535 different service types. The maximum payload and a payload length up to 100 bytes. Despite this, most DCP messages are expected to be very small.

4.7 Packet Size

The maximum size of a DCP packet is 100 bytes. The DCP header is 4 bytes long, leaving up to 96 bytes for the DCP payload.

A challenge while developing the DCP was to define a maximum DCP packet size that would cover as many transmission technologies as possible without requiring a mandatory fragmentation and reassembly mechanism, and without defining the maximum packet size too small for certain home automation scenarios. Various underlying transmission technologies and communication protocols differ strongly in their definition of the maximum transmission unit (MTU). An MTU of an underlying layer is the maximum number of bytes that the layer can accept from the DCP layer and transfer at a time. The MTU size directly limits the maximum size of a DCP packet that can be transferred, as the DCP packet can not be larger than the MTU of the underlying technology.

The initial solution was actually not to define a maximum DCP packet size that all implementations would comply with. The theoretical maximum packet size of the DCP packet was 65536 bytes, and the actual maximum packet size would have to be adopted to the MTU of the underlying transmission technology by each DCP implementation. The initial solution was later abandoned, primarily because it introduced packet size variations making it harder to implement interoperatible products.

The final solution defines the maximum DCP packet size to 100 bytes. This limit is highly affected by the MTU of the IEEE 802.15.4 MAC layer being 102 bytes [LRW03]. It was considered very important to support the IEEE 802.15.4 as a DCP transmission technology, the only globally standardized short-range wireless technology optimized for home automation and similar areas. A maximum packet size that all DCP implementations must offer has been specified (at the expense of the initial solution) primarily in order to ensure the interoperability between DCP appliances.

Most DCP packets are expected to be very small, ranging from 4-30 bytes. Therefore, the DCP packet size of maximum 100 bytes does not represent a restriction regarding the DCP application area range.

4.8 Message Types

This section specifies the DCP message types. The messages are listed in Table 4.1. Each message type is defined separately in following subsections. Representation primitives used throughout the section are defined in Table 4.2.

Message name	Message number
CONNECT_REQ	0x00
CONNECT_RSP	0x01
CONNECT_ERR	0x02
DISCONNECT_REQ	0x03
DISCONNECT_RSP	0x04
DISCONNECT_ERR	0x05
BIND_REQ	0x06
BIND_RSP	0x07
BIND_ERR	0x08
UNBIND_REQ	0x09
UNBIND_RSP	0x0a
UNBIND_ERR	0x0b
SETDATA_REQ	0x0c
SETDATA_RSP	0x0d
SETDATA_ERR	0x0e
GETDATA_REQ	0x0f
GETDATA_RSP	0x10
GETDATA_ERR	0x11
SERVICE_DISCOVERY_REQ	0x12
SERVICE_DISCOVERY_RSP	0x13
SERVICE_DISCOVERY_ERR	0x14
DEVICE_DESCRIPTION_REQ	0x15
DEVICE_DESCRIPTION_RSP	0x16
DEVICE_DESCRIPTION_ERR	0x17

Table 4.1: DCP message types

Primitive type	Number of bytes	Maximum range
Unsigned byte	1	0...255
Signed byte	1	-128...127
Unsigned integer	2	0...65535
Signed integer	2	-32768...32767

Table 4.2: Representation primitives

4.8.1 CONNECT_REQ

This message requests a connection at the DCP layer. A DCP connection must exist prior to exchanging any other messages. The `CONNECT_REQ` payload is defined in Table 4.3. The payload contains the version of the protocol. The current version of the protocol is 1, and will be incremented by 1 for each new version.

Primitive type	Primitive name	Valid range
Unsigned byte	Protocol version	0x01...0xff

Table 4.3: CONNECT_REQ payload

4.8.2 CONNECT_RSP

This message informs the receiver that the remote device has accepted the DCP connection. The `CONNECT_RSP` payload is defined in Table 4.4. The payload contains the version of the protocol. The current version of the protocol is 1, and will be incremented by 1 for each new version.

Primitive type	Primitive name	Valid range
Unsigned byte	Protocol version	0x01...0xff

Table 4.4: CONNECT_RSP payload

4.8.3 CONNECT_ERR

If a device does not accept the DCP connection request, it shall return a `CONNECT_ERR` message. The `CONNECT_ERR` payload is defined in Table 4.5. The payload shall contain an error reason.

Primitive type	Primitive name	Valid range
Unsigned byte	Error reason	CONNECTION_REJECTED CONNECTION_EXISTS CONNECTION_OVERFLOW UNSUPPORTED_VERSION

Table 4.5: CONNECT_ERR payload

4.8.4 DISCONNECT_REQ

This message tells the remote device to close the current DCP connection and the connection at the layer below the DCP. The payload length field shall be set to 0x00, and the payload shall not be included in the message.

4.8.5 DISCONNECT_RSP

This message informs that the state information that was associated with the connection has been deleted. It is the responsibility of the device that initiated the disconnect procedure by sending the `DISCONNECT_REQ` message to close the physical connection at the layer below DCP. The payload length field shall be set to 0, and the payload shall not be included in the message.

4.8.6 DISCONNECT_ERR

If a device is not able to close the connection, it should respond with a `DISCONNECT_ERR` message. The `DISCONNECT_ERR` payload is defined in Table 4.6. The payload shall contain the error reason.

Primitive type	Primitive name	Valid range
Unsigned byte	Error reason	NO_CONNECTION

Table 4.6: DISCONNECT_ERR payload

4.8.7 BIND_REQ

This message is used to request a binding between two services on two different devices. The `BIND_REQ` payload is defined in Table 4.7.

The first two bytes represent the service type. ServiceType specifies the service to be bound. Bind direction is the direction of the binding, it should either be `BIND_IN_OUT`, `BIND_IN`, or `BIND_OUT`. The direction is always considered from the viewpoint of the device initiating the binding (i.e. the device that sends `BIND_REQ` message). If the direction is set to `BIND_IN`, the device is expected to be passive, it will receive `SETDATA_REQ` or `GETDATA_REQ` messages and only respond to these. If the direction is set to `BIND_OUT`, the device will be active and control the remote service by sending `SETDATA_REQ` or `GETDATA_REQ` messages. If the direction is set to `BIND_IN_OUT`, then the device is free to send all types of messages.

Each service implemented on a device has an associated service port number. Service port number is a simple integer functioning as a service multiplexer in scenarios where a device implements multiple instances of the same service type (i.e. light control panel with multiple switches). The service port number must be unique within the device, meaning that the service port number can uniquely address a service instance within a device. The remote service port number can be discovered in a service discovery procedure. It is also possible to set the remote service port number is set to `PORT_ANY`, making the service port number unspecified and letting the remote device choose any free service port number for the specified service type.

Local port is the port number of the device that sent the `BIND_REQ` message. Remote port is the port number of the device that receives the `BIND_REQ` message.

Primitive type	**Primitive name**	**Valid range**
Unsigned integer	Service type	0x10...0xffff
Unsigned byte	Bind direction	BIND_IN_OUT = 0x00, BIND_IN = 0x01, BIND_OUT = 0x02
Unsigned byte	Local port	0x01...0xff
Unsigned byte	Remote port	PORT_ANY = 0x00, 0x01...0xff

Table 4.7: BIND_REQ payload

4.8.8 BIND_RSP

This message tells that the remote device has accepted the binding request. The `BIND_RSP` payload is defined in Table 4.8. Service type must be set to the same value as in `BIND_REQ` message. The bind direction must equal the corresponding field in the request message. If the port in the request message was set to `PORT_ANY`, the device can use any free port for the specified service. In this message, local port is the local port of the device that sent the `BIND_REQ` message, and remote port is the port of the device that transmits the `BIND_RSP`.

Primitive type	Primitive name	Valid range
Unsigned integer	Service type	0x10...0xffff
Unsigned byte	Bind direction	BIND_IN_OUT = 0x00, BIND_IN = 0x01, BIND_OUT = 0x02
Unsigned byte	Local port	0x01...0xff
Unsigned byte	Remote port	0x01...0xff

Table 4.8: BIND_RSP payload

4.8.9 BIND_ERR

If a device does not accept the binding request, it shall respond with a `BIND_ERR` message. The `BIND_ERR` payload is defined in Table 4.9. The payload shall contain the error reason.

Primitive type	Primitive name	Valid range
Unsigned byte	Error reason	INCORRECT_VALUE, INVALID_SERVICE, BINDING_REJECTED, BINDING_EXISTS, BINDING_OVERFLOW, INVALID_PORT, NO_CONNECTION

Table 4.9: BIND_ERR payload

4.8.10 UNBIND_REQ

This message asks the receiver to remove the specified binding. The `UNBIND_REQ` payload is defined in Table 4.10. Service type must be set to service we want to unbind. The payload contains the direction of the binding, the local port and the remote port.

Primitive type	Primitive name	Valid range
Unsigned integer	Service type	0x10...0xffff
Unsigned byte	Bind direction	BIND_IN_OUT = 0x00, BIND_IN = 0x01, BIND_OUT = 0x02
Unsigned byte	Local port	0x01...0xff
Unsigned byte	Remote port	0x01...0xff

Table 4.10: UNBIND_REQ payload

4.8.11 UNBIND_RSP

This message confirms that the binding has successfully been removed. The `UNBIND_RSP` payload is defined in Table 4.11. Service type shall be the same as in `UNBIND_REQ`. The bind direction must equal the corresponding field in the request message. The local port is the port of the device that sent the `BIND_REQ` message, and remote port is the port of the device that transmits the `BIND_RSP`.

Primitive type	Primitive name	Valid range
Unsigned integer	Service type	0x10...0xffff
Unsigned byte	Bind direction	BIND_IN_OUT = 0x00, BIND_IN = 0x01, BIND_OUT = 0x02
Unsigned byte	Local port	0x01...0xff
Unsigned byte	Remote port	0x01...0xff

Table 4.11: UNBIND_RSP payload

4.8.12 UNBIND_ERR

This message informs the receiver that the sender was unable to remove the binding. The `UNBIND_ERR` payload is defined in Table 4.12. The payload shall contain the error reason.

Primitive type	Primitive name	Valid range
Unsigned byte	Error reason	NO_BINDING, INVALID_PORT, NO_CONNECTION

Table 4.12: UNBIND_ERR payload

4.8.13 SETDATA_REQ

This message tells the receiver to change the value of the specified service to what is specified in the Payload field. The `SETDATA_REQ` payload is defined in Table 4.13. This message shall only be sent if a valid binding for the specified service type exists. The payload defines the specified the binding through which the message is sent, and the new value of the specified service. The service value representation and formatting is dependent on the service type field.

Primitive type	Primitive name	Valid range
Unsigned integer	Service type	0x10...0xffff
Unsigned byte	Local port	0x01...0xff
Unsigned byte	Remote port	0x01...0xff
Defined by Service type	Service value	Defined by Service type

Table 4.13: SETDATA_REQ payload

4.8.14 SETDATA_RSP

This message tells the receiver that the request to change a service value was successful. The `SETDATA_RSP` payload is defined in Table 4.14. Service type shall be the same as in `SETDATA_REQ` message. The service value is dependent on the service type field.

Primitive type	Primitive name	Valid range
Unsigned integer	Service type	0x10...0xffff
Unsigned byte	Local port	0x01...0xff
Unsigned byte	Remote port	0x01...0xff
Defined by Service type	Service value	Defined by Service type

Table 4.14: SETDATA_RSP payload

4.8.15 SETDATA_ERR

This message informs the receiver that the sender did not change the value of the service specified in `SETDATA_REQ`. The `SETDATA_ERR` payload is defined in Table 4.15. The payload shall contain the error reason.

Primitive type	Primitive name	Valid range
Unsigned byte	Error reason	NO_CONNECTION, NO_BINDING, INCORRECT_VALUE, INVALID_PORT, INCORRECT_DIRECTION, PACKET_TOO_LARGE

Table 4.15: SETDATA_ERR payload

4.8.16 GETDATA_REQ

This message is used to request the current value of the specified service. The value of the service is sent back to the requesting device in the `GETDATA_RSP` message. The `GETDATA_REQ` payload is defined in Table 4.16. The service is identified by setting the service type field to the service port number agreed upon during the binding procedure. This message shall only be sent if a valid binding for the specified service exists. The payload defines the binding through which the message is sent, and the new value of the specified service. The service value is dependent of the service type field.

Primitive type	Primitive name	Valid range
Unsigned integer	Service type	0x10...0xffff
Unsigned byte	Local port	0x01...0xff
Unsigned byte	Remote port	0x01...0xff
Defined by Service type	Service value	Defined by Service type

Table 4.16: GETDATA_REQ payload

4.8.17 GETDATA_RSP

This message carries the value of the service that was requested in `GETDATA_REQ`. The `GETDATA_RSP` payload is defined in Table 4.17. Service type is set to the service port number of the current service. The payload defines the binding through which the message is sent, and the new value of the specified service. The service value is dependent of the service type field.

Primitive type	Primitive name	Valid range
Unsigned integer	Service type	0x10...0xffff
Unsigned byte	Local port	0x01...0xff
Unsigned byte	Remote port	0x01...0xff
Defined by Service type	Service value	Defined by Service type

Table 4.17: GETDATA_RSP payload

4.8.18 GETDATA_ERR

This message tells the receiver that the sender could not send the current value of the service specified in `GETDATA_REQ`. The `GETDATA_ERR` payload is defined in Table 4.18. The payload shall contain the error reason.

4.8.19 SERVICE_DISCOVERY_REQ

This message is used to perform a service discovery and find out which service types a remote device implements. The `SERVICE_DISCOVERY_REQ` payload is defined in Table 4.19. If the service type field is set to `ALL_SERVICES`, a list of all services is returned to the requesting device. If the service type field is set to a specific service type, then the discovery applies only to the specified service.

Primitive type	Primitive name	Valid range
Unsigned byte	Error reason	NO_CONNECTION, NO_BINDING, INCORRECT_VALUE, INVALID_PORT, INCORRECT_DIRECTION, PACKET_TOO_LARGE

Table 4.18: GETDATA_ERR payload

Primitive type	Primitive name	Valid range
Unsigned integer	Service type	ALL_SERVICES = 0x00, 0x10...0xffff

Table 4.19: SERVICE_DISCOVERY_REQ payload

4.8.20 SERVICE_DISCOVERY_RSP

This message carries the results of a service discovery. The `SERVICE_DISCOVERY_RSP` payload is defined in Table 4.20. Service type should be set to the same value as in `SERVICE_DISCOVERY_REQ`. Service count tells how many services follow in the list. The maximum number of service types to be contained in the list is 255. Service type, local port, remote port and the direction of the service are repeated sequentially for each service on the device.

4.8.21 SERVICE_DISCOVERY_ERR

This message signalizes an error in the service discovery procedure. The `SERVICE_DISCOVERY_ERR` is defined in Table 4.21. The payload shall contain the error reason.

4.8.22 DEVICE_DESCRIPTION_REQ

This message requests a device description from the remote device. The Payload Length field shall be set to 0, and payload shall not be included in this message.

Primitive type	Primitive name	Valid range
Unsigned byte	Service count	0x00...0xff
Unsigned integer	Service type	0x10...0xffff
Unsigned byte	Local port	0x01...0xff
Unsigned byte	Remote port	0x01...0xff
Unsigned byte	Direction	BIND_IN_OUT = 0x00, BIND_IN = 0x01, BIND_OUT = 0x02
...	...	...
Unsigned integer	Service type	0x10...0xffff
Unsigned byte	Local port	0x01...0xff
Unsigned byte	Remote port	0x01...0xff
Unsigned byte	Direction	BIND_IN_OUT = 0x00, BIND_IN = 0x01, BIND_OUT = 0x02

Table 4.20: SERVICE_DISCOVERY_RSP payload

Primitive type	Primitive name	Valid range
Unsigned byte	Error reason	NO_CONNECTION PACKET_TOO_LARGE

Table 4.21: SERVICE_DISCOVERY_ERR payload

4.8.23 DEVICE_DESCRIPTION_RSP

This message carries the device description. The `DEVICE_DESCRIPTION_RSP` payload is defined in Table 4.22. Device name is a user-friendly name such as "Switch" or "Thermostat". Manufacturer name is a user-friendly name of the device manufacturer. Device model is a user-friendly description of the device model.

Primitive type	Number of bytes	Primitive name	Valid range
Unsigned byte	1	Device name length	0x00...0x14
Unsigned byte	Device name length	Device name	Each byte represents a character of the name according to ASCII table
Unsigned byte	1	Manufacturer name length	0x00...0x14
Unsigned byte	Manufacturer name length	Manufacturer name	Each byte represents a character of the name according to ASCII table
Unsigned byte	1	Device model length	0x00...0x14
Unsigned byte	Device model length	Device model	Each byte represents a character of the name according to ASCII table

Table 4.22: DEVICE_DESCRIPTION_RSP payload

4.8.24 DEVICE_DESCRIPTION_ERR

This message signalizes that the device could not generate and send a device description to the requesting device. The `DEVICE_DISCOVERY_ERR` is defined in Table 4.23. The payload shall contain the error reason.

Primitive type	Primitive name	Valid range
Unsigned byte	Error reason	NO_CONNECTION PACKET_TOO_LARGE

Table 4.23: DEVICE_DESCRIPTION_ERR payload

4.9 DCP Services

The standardized DCP service types are listed in Table 4.24 and described individually in following subsections. Service type numbers from 0x8000 to 0xffff are reserved for manufacturers to define their own proprietary service types.

Service name	Service number
SERVICE_DATE	0x10
SERVICE_TIME	0x11
SERVICE_SWITCH	0x12
SERVICE_DIMMER	0x13
SERVICE_TEMP_C	0x14

Table 4.24: DCP service types

4.9.1 SERVICE_DATE

This service represents the date on format day-month-year. The `SERVICE_DATE` structure is defined in Table 4.25.

Primitive type	Primitive name	Valid range
Unsigned byte	Day	0x01...0x1f
Unsigned byte	Month	0x01...0x0c
Unsigned integer	Year	0x00...0xffff

Table 4.25: SERVICE_DATE structure

4.9.2 SERVICE_TIME

This service represents the time on format hours-minutes-seconds. The `SERVICE_TIME` structure is defined in Table 4.26.

4.9.3 SERVICE_SWITCH

This service represents an on/off state. The `SERVICE_SWITCH` structure is defined in Table 4.27.

Primitive type	Primitive name	Valid range
Unsigned byte	Hours	0x00...0x17
Unsigned byte	Minutes	0x00...0x3b
Unsigned byte	Seconds	0x00...0x3b

Table 4.26: SERVICE_TIME structure

Primitive type	Primitive name	Valid range
Unsigned byte	Switch	SWITCH_OFF = 0x00, SWITCH_ON = 0x01

Table 4.27: SERVICE_SWITCH structure

4.9.4 SERVICE_DIMMER

This service represents the percentage level of the current dimmer value. The `SERVICE_DIMMER` structure is defined in Table 4.28. The minimum value of the dimmer is 0x00 (light is completely off). The maximum value of the dimmer is 0xff (maximum light intensity).

Primitive type	Primitive name	Valid range
Unsigned byte	Dimmer	0x00...0xff

Table 4.28: SERVICE_DIMMER structure

4.9.5 SERVICE_TEMP_C

This service represents the temperature in degrees of Celsius. The resolution is one degree. The `SERVICE_TEMP_C` structure is defined in Table 4.29.

4.10 Error reasons

If a device is unable to carry through the job specified in a request message (with suffix `REQ`), it should return an error message (with suffix `ERR`). The sequence number must always equal the value in the corresponding request message. Payload shall contain the error reason. DCP error reasons are defined in following paragraphs. The error reason number sent on air is defined in the parenthesis following the error reason name.

Primitive type	Primitive name	Valid range
Signed byte	Temperature in Celsius degrees	-125...125

Table 4.29: SERVICE_TEMP_C structure

NO_BINDING (0x10) The request could not be carried through because there is no valid binding for the requested service.

INCORRECT_VALUE (0x11) The service value is incorrect.

NO_CONNECTION (0x12) The request could not be carried through because there is no valid connection to the requesting device.

NO_SERVICES (0x13) The requested operation could not be completed because the device does not implement the requested service type(s).

CONNECTION_REJECTED (0x14) The incoming connection request could not be accepted.

CONNECTION_EXISTS (0x15) Connection already exists.

CONNECTION_OVERFLOW (0x16) Connection could not be created because a maximum number of active connections have already been reached.

UNSUPPORTED_VERSION (0x17) Requested protocol version is not supported.

INVALID_SERVICE (0x18) The service that was specified in the request message is no valid.

BINDING_REJECTED (0x19) The incoming binding request could not be accepted.

BINDING_EXISTS (0x1a) A valid binding for the specified parameters already exists.

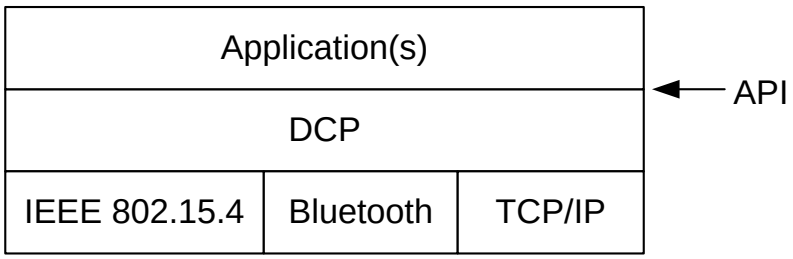

Figure 4.6: Device Control Protocol (DCP) application program interface (API)

BINDING_OVERFLOW (0x1b) Binding could not be created because a maximum number of active bindings have already been reached.

INCORRECT_DIRECTION (0x1c) The operation could not be completed because the direction of the binding is incorrect for this request.

INVALID_PORT (0x1d) The specified port is incorrect.

PACKET_TOO_LARGE (0x1e) The size of the DCP packet is too large and it could not be transmitted or received.

4.11 Application Program Interface (API)

This section proposes a standardized, generic application program interface (API) for the Device Control Protocol (DCP) that has been developed and implemented in various prototype systems during the thesis. The DCP API hides the complex details of the underlying transmission technologies provides a simple, uniform programming interface. The API is completely independent of the underlying technology, which means that the API is the same no matter which underlying transmission technology the DCP is implemented upon. The transmission technology can be replaced without affecting the application since the changes only must be made at the DCP layer, while the interface the application sees remains unchanged. Figure 4.6 illustrates the DCP API.

All over-the-air communication is based on request-response transactions. If a device sends a request command, it expects to either receive a response message or a corresponding error message. This principle is exploited in the DCP implementation. An application makes a call to a request function, which creates a DCP message, transmits it and waits for either a response or an error message. The success or the failure of the transaction is indicated to the application through the return value of the request function. If the function waiting for the response times out, it should return an error value back to application.

The receiver is obligated to execute a callback to a correct request handler function provided by the application. These request handler functions are called indication functions, as they indicate that a request message has been received and it needs to be handled. An indication function is required to answer to the request, either with a response or an error message by calling the appropriate function at the DCP layer. The DCP API implementation architecture is illustrated in Figure 4.7. Note that the suggested architecture assumes that a request function waits and does not return until the transaction with a remote device has been completed, or until a timeout occurs in which case the error is indicated to the application through the function return value.

Request function names have the suffix Req (e.g. `dcpConnectReq`), indication functions have the suffix Ind (e.g. `dcpConnectInd`), response functions have the suffix Rsp (e.g. `dcpConnectRsp`) while error functions have the suffix Err (e.g. `dcpConnectErr`).

The API functions names are generated by adding the prefix "dcp" to the DCP message name and then eliminating the underscore separator "_". For example, the API function that creates a connection to a remote device is called `dcpConnectReq`, while the corresponding DCP message actually sent on air is called `CONNECT_REQ`. The syntax of the API function calls in the following subsections is similar to the C programming language.

The sequence diagrams in sections below use terms DCP client and DCP server. DCP client is a DCP layer of the device that sends out a DCP request message (e.g. `BIND_REQ`) thus starting a DCP request-response transaction. A device that receives a DCP request message is called DCP server.

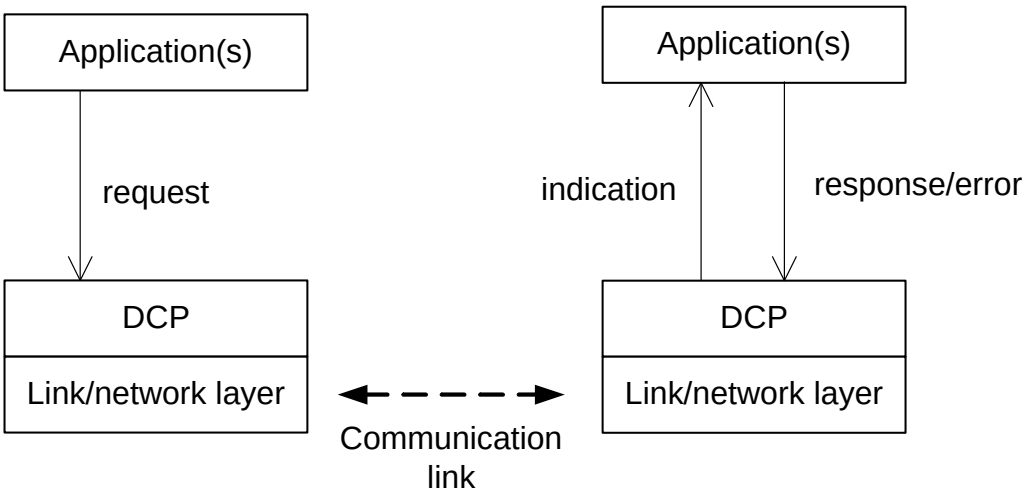

Figure 4.7: DCP API implementation architecture

4.11.1 Scanning for Devices

The application can discover other devices in the transmission range of the technology the DCP implementation is based on by calling the function `dcpScanReq`. The device scan procedure is illustrated in Figure 4.8. The implementation of the `dcpScanReq` function is responsible for performing the device scan using the capabilities of the underlying technology. If the technology does not offer device scan mechanism, then this subsection should be ignored by the specific implementation.

The function will perform the device scan, and for each device found it will produce a 16-bit DCP address (see Section 4.3). The application must provide a data structure (illustrated as "neighbors" in Figure 4.8) where the results of the scan (16-bit DCP addresses) will be put.

4.11.2 Connecting

A DCP connection between two devices is established by calling the `dcpConnectReq` function. The connect procedure is illustrated in Figure 4.9. The application has to provide a valid 16-bit DCP address of the remote device to the function (illustrated as "serverAddress" in Figure 4.9). The application can acquire the DCP addresses of the neighboring devices prior to calling `dcpConnectReq`

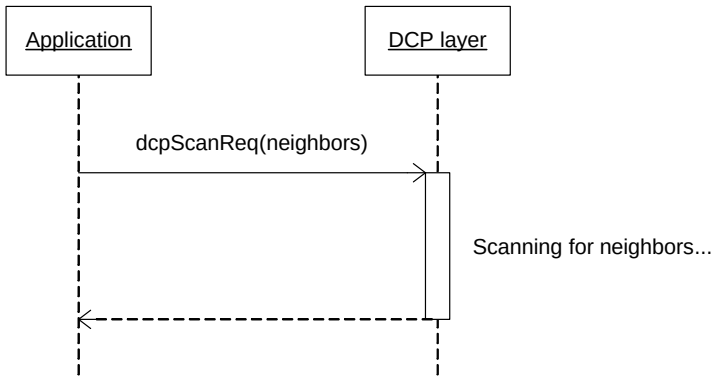

Figure 4.8: Device scan procedure

by calling the `dcpScanReq` function (see Section 4.11.1).

The `dcpConnectReq` function creates a `CONNECT_REQ` message and transmits it. The receiving DCP entity executes a callback to dcpConnectInd function, sending the locally generated 16-bit DCP address procedure as a parameter (illustrated as "clientAddress" in Figure 4.9). The `dcpConnectInd` function is responsible for answering the DCP client by calling either the `dcpConnectRsp` function if the connection request is accepted, or the `dcpConnectErr` function if the connection can not be accepted.

4.11.3 Disconnecting

A DCP connection between two devices is closed by calling the `dcpDisconnectReq` function. The disconnect procedure is illustrated in Figure 4.10. The application has to provide a valid DCP address of the remote device (illustrated as "serverAddress" in Figure 4.10).

The function creates a `DISCONNECT_REQ` message and transmits it. The receiving DCP server executes a callback to the `dcpDisconnectInd` function, sending the locally generated DCP address of that started the disconnect procedure as a parameter (illustrated as "clientAddress" in Figure 4.10). The `dcpDisconnect` function is responsible for answering the DCP client by calling either the `dcpDisconnectRsp` function, or the `dcpDisconnectErr` function if the connection can not be closed.

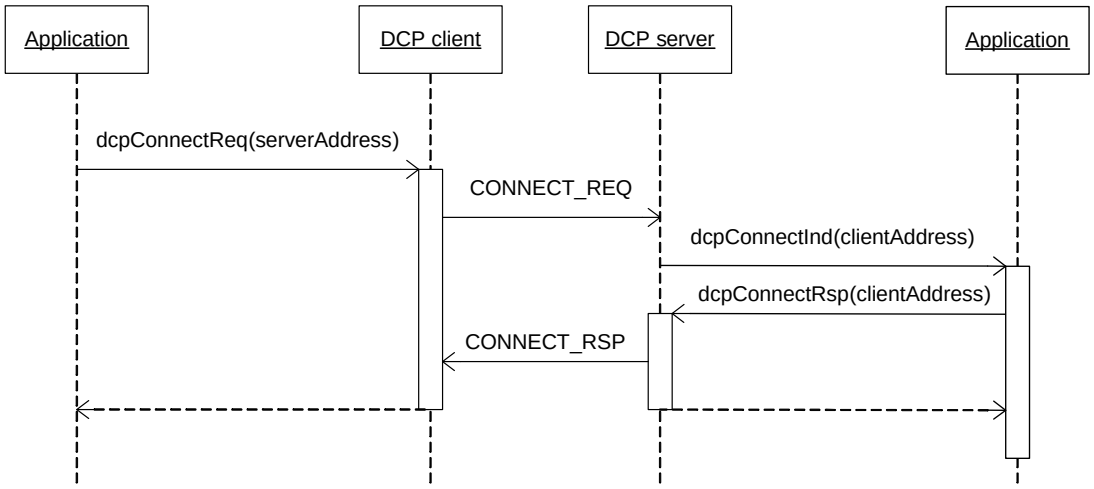

Figure 4.9: Connect procedure

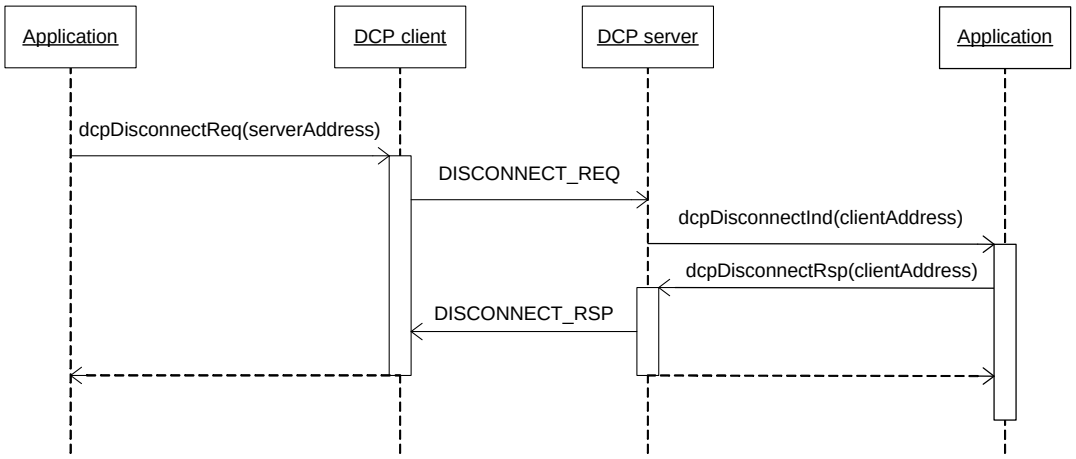

Figure 4.10: Disconnect procedure

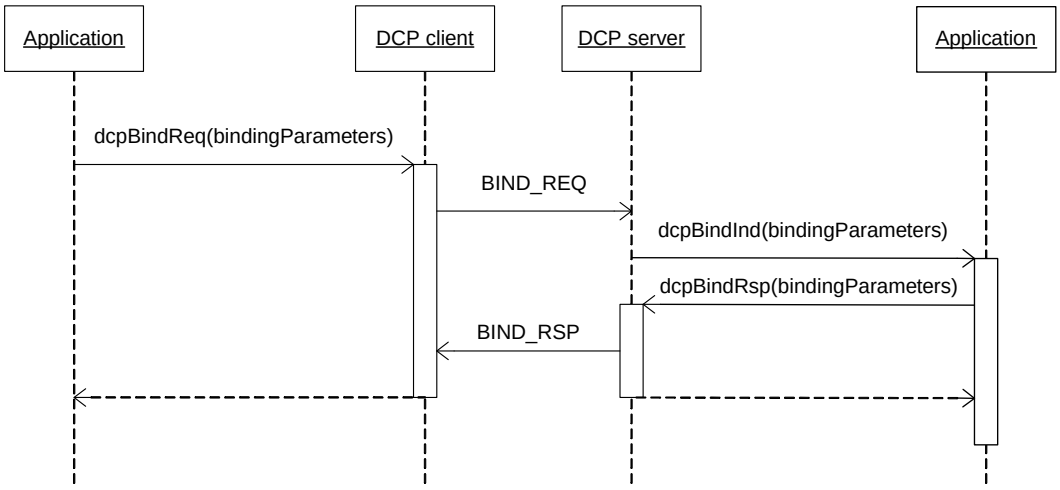

Figure 4.11: Bind procedure

4.11.4 Binding

A DCP binding between two services is created by calling the `dcpBindReq` function. The bind procedure is illustrated in Figure 4.11. The function parameters must uniquely describe the binding: DCP address of the remote device, service type, local port, remote port and binding direction. The function parameters are illustrated as "bindingParameters" in Figure 4.11. Using these parameters, the function assembles a `BIND_REQ` message and transmits it. The receiving DCP server executes a callback to `dcpBindInd` function. If the device accepts the binding request, it answers the DCP client by calling either the `dcpBindRsp` function. If the binding request is rejected, it calls the `dcpBindErr` function which assembles and transmits a `BIND_ERR` message.

4.11.5 Unbinding

A DCP binding between two services is removed by calling the `dcpUnbindReq` function. The unbind procedure is illustrated in Figure 4.12. The function parameters must uniquely describe the binding to be removed: the DCP address of the remote device, service type, local port, remote port and binding direction. The function parameters are illustrated as "bindingParameters" in

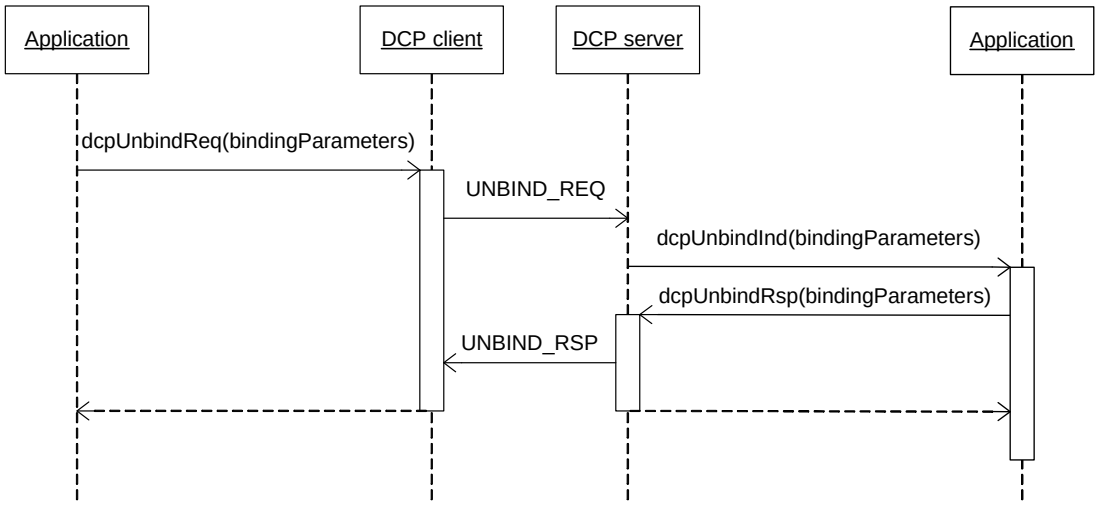

Figure 4.12: Unbind procedure

Figure 4.12. Using these, the function assembles a `UNBIND_REQ` message and transmits it. The receiving DCP server executes a callback to `dcpUnbindInd` function, which is responsible for answering the DCP client by calling either the `dcpUnbindRsp` function if the binding was deleted or the `dcpUnbindErr` if the binding could not be removed.

4.11.6 Changing the Service Value

An application can change the value of a service on a remote device by calling the `dcpSetDataReq` function. The procedure of changing the service value is illustrated in Figure 4.13. It is a responsibility of the application to ensure that a valid binding exists prior to calling the `dcpSetDataReq` function. The application must provide several parameters to the function that uniquely describe the binding (DCP address of the remote device, service type, local port, remote port and binding direction), illustrated as "bindingParameters" in Figure 4.13. The application must also provide the data to be sent (illustrated as "data"in Figure 4.13).

The function assembles a `SETDATA_REQ` message and transmits it. The receiving DCP server executes a callback to `dcpSetDataInd` function, which is responsible for actually changing the value of the specified service. The

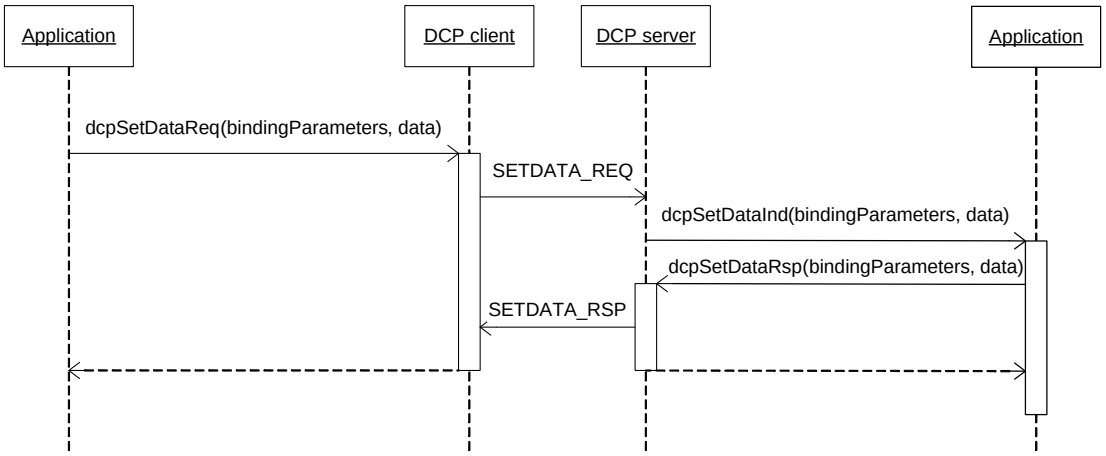

Figure 4.13: Changing the service value

`dcpSetDataInd` answers the DCP client by calling either the `dcpSetDataRsp` function in case of success or `dcpSetDataErr` if the service value could not be changed.

4.11.7 Reading the Service Value

An application can read the value of a service on a remote device by calling the `dcpDeviceDescriptionReq` function. The procedure of reading the value of a service is illustrated in Figure 4.14. It is a responsibility of the application to ensure that a valid binding exists prior to calling the `dcpGetDataReq` function. The application must provide several function parameters (illustrated as "bindingParameters" in Figure 4.14) that uniquely describe the binding (DCP address of the remote device, service type, local port, remote port and binding direction). The application must also provide a parameter (illustrated as "data" in Figure 4.14) where the data received from the remote device will be put by the function.

The function assembles a `GETDATA_REQ` message and transmits it. The receiving DCP server executes a callback to `dcpGetDataInd` function, which answers the DCP client by calling either the `dcpGetDataRsp` function specifying the current service value or `dcpGetDataErr` in case of an error.

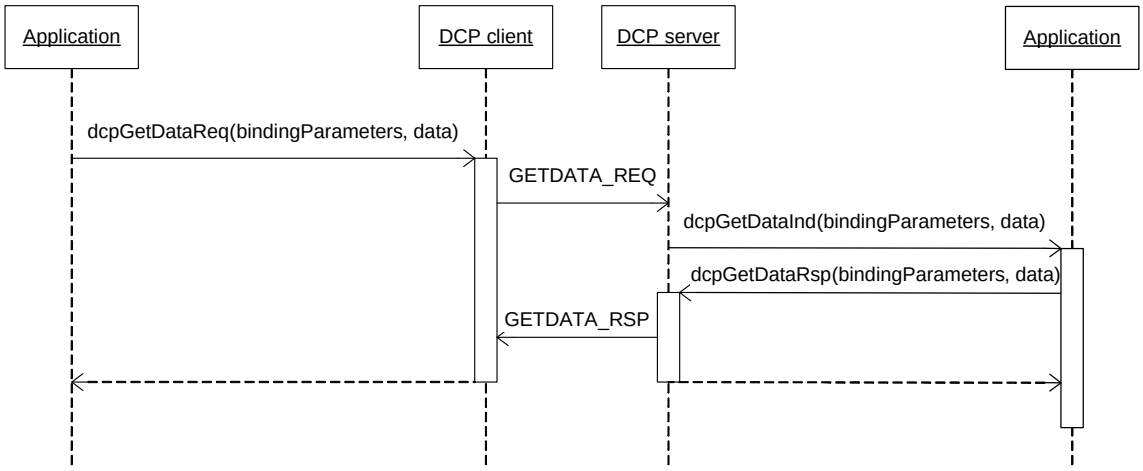

Figure 4.14: Reading the service value

4.11.8 Service Discovery

An application can discover what services some other device offers by calling the `dcpServiceDiscoveryReq` function. The service discovery procedure is illustrated in Figure 4.15. The application has to provide a valid DCP address of the remote device (illustrated as "address" in 4.15). The application can either search for all available services or a specific service, as described in Section 4.8.19. The service search type is illustrated as "service" in Figure 4.15. The application also has to provide a data structure (illustrated as “result” in Figure 4.15) where the results of the service discovery will be put. The `dcpServiceDiscoveryReq` function assembles a `SERVICE_DISCOVERY_REQ` message and transmits it. The receiving DCP server executes a callback to the `dcpServiceDiscoveryInd` function, which is responsible for answering the DCP client by calling either the `dcpServiceDiscoveryRsp` function specifying the discovery results, or the `dcpServiceDiscoveryErr` function if the discovery could not be carried through.

4.11.9 Device Description

An application can request a device description from some other device by calling the `dcpDeviceDescriptionReq` function. The device description procedure is illustrated in Figure 4.16. The application has to pro-

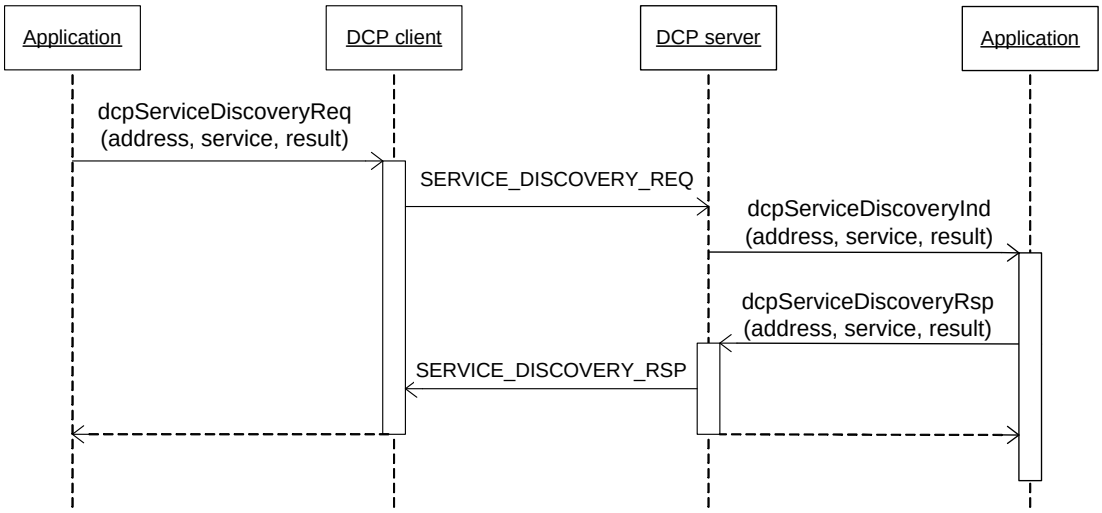

Figure 4.15: Service discovery procedure

vide a valid DCP address of the remote device (illustrated as "address" in 4.16). The application also has to provide a data structure (illustrated as "description"in Figure 4.16) where the results of the device description search will be put. The `dcpDeviceDescriptionReq` function assembles a `DEVICE_DESCRIPTION_REQ` message and transmits it. The receiving DCP server executes a callback to `dcpDeviceDescriptionInd` function, which is responsible for answering the DCP client by calling either the `dcpDeviceDescriptionRsp` function specifying it's description, or the `dcpDeviceDescriptionErr` function in case of an error.

4.12 DCP Bridging

DCP can be implemented upon various transmission technologies, and it could potentially implement a bridging mechanism to glue together DCP networks based on different transmission technologies. Although not a part of the DCP specification (primarily due to lack of time), a DCP bridging functionality would be a very useful service. The scenario is illustrated in Figure 4.17, where an Internet remote device (i.e. web site) controls a home appliance through the DCP gateway (bridge). Some issues that should be considered in this context:

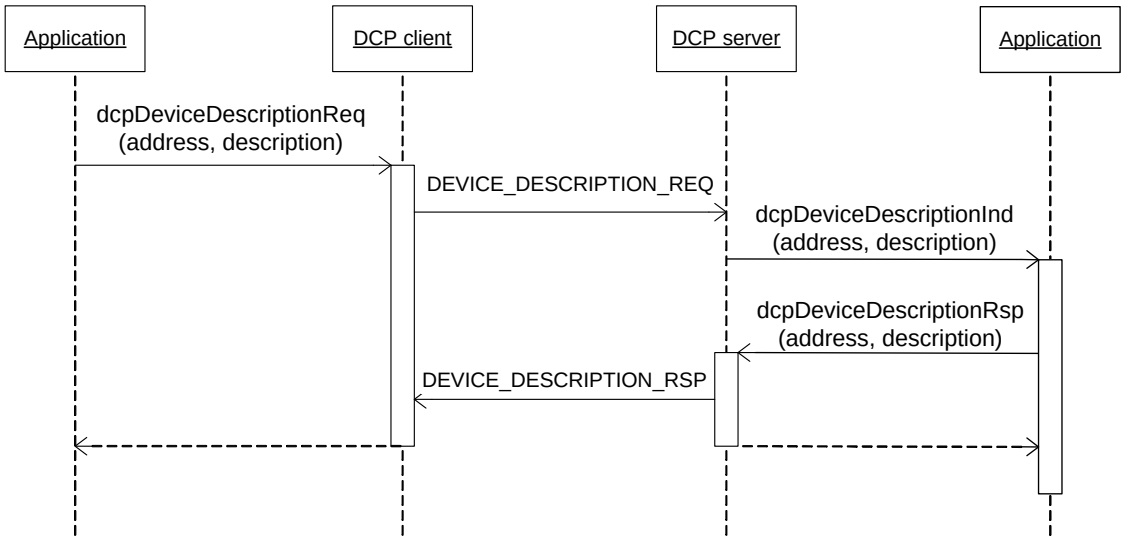

Figure 4.16: Device description procedure

- Device addressing
- Additional DCP messages
- Communication security

4.13 Network Layer

This section is a short discussion of routing protocols in the context of IEEE 802.15.4 technology and the Device Control Protocol (DCP). The theoretical investigation of the thesis concluded that IEEE 802.15.4 is the technology most suitable for wireless home automation networking. The goal of the section is to identify the desirable properties of a routing mechanism and outline a routing strategy for IEEE 802.15.4 networks running the DCP protocol. In a protocol stack, the routing mechanism would be implemented upon the IEEE 802.15.4 technology but below DCP, as illustrated in Figure 4.18.

DCP relies on the underlying layer to provide a sufficient transmission range to suit the application needs, but in some scenarios this simply is not possible unless multihop communication is used. By introducing a network layer and a routing mechanism at the layer below DCP, the transmission

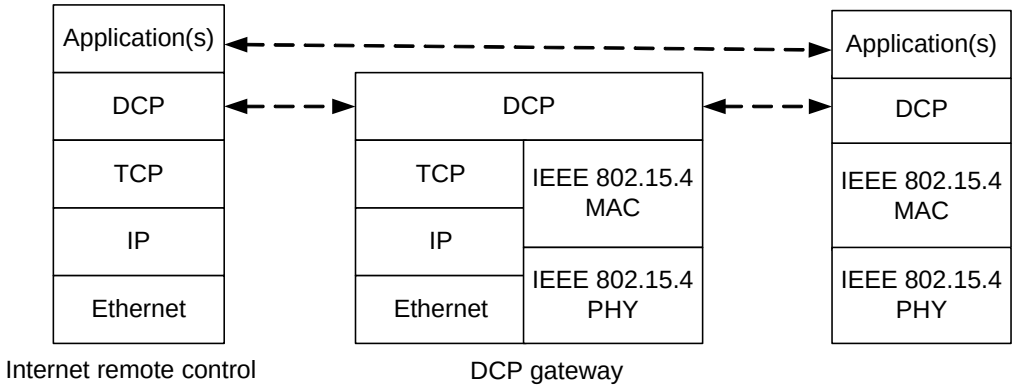

Figure 4.17: DCP bridging scenario

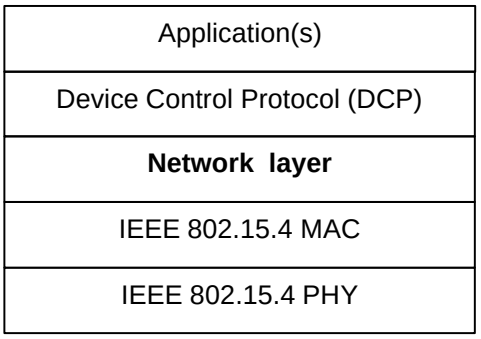

Figure 4.18: Position of network layer in a DCP stack

range could be increased without increasing the transmission output power or the power consumption. In such a case, some DCP devices would function as mobile routers and they would forward packets toward their correct destination.

The IEEE 802.15.4 is still a relatively new technology, the final specification has been released in 2003. Up to now, a small amount of academic research efforts has focused on the technology. The research efforts are primarily carried out by the industry, with the ZigBee Alliance [ZIG] leading the way. This is likely to change, as the popularity of the IEEE 802.15.4 and the related ZigBee technology is forecast to increase in the years to come.

The IEEE Mobile Ad-hoc Networks (MANET) [IET] working group has set the goal to standardize IP routing protocol functionality suitable for wireless routing within both static and dynamic topologies. The MANET group has standardized several routing protocols [IET]with different properties, purposes and areas of application. Several other protocols have been defined outside the MANET group. An overall review of the ad hoc routing protocols can be found in [Mis99] and [Roy99]. None of these protocols are designed specifically for the IEEE 802.15.4 technology. However, the principles and the main ideas should be reusable and adaptable to the IEEE 802.15.4 technology.

The IEEE has started the IEEE 802.15.5 Task Group [WPAb] with the main objective of determining the necessary mechanisms that must be present in the PHY and MAC layers of the short-range wireless technology to enable mesh networking, and to provide recommendations for building of mesh networks. The working group will try to find out how mesh networks can be formed at the MAC layer, without needing ZigBee or some other routing protocol.

4.13.1 Reactive vs. Proactive Protocols

IEEE 802.15.4 peer-to-peer network topology supports many different types of networks, and the designer must select the appropriate network type and the routing algorithm for the intended application. Important tasks of the protocol are to minimize battery power consumption and use of bandwidth and CPU and memory resources. Routing protocols may generally be categorized as [Roy99]:

- Proactive (Table-driven)

- Reactive (Source routing)

Proactive routing protocols maintain consistent routes from each node to every other node in the network. Routing information is stored in one on more tables. Changes in network topology are handled by sending updates throughout the network. Proactive protocols maintain a route entry for every other network node, even if they never talk to many of these. This requires unnecessary memory and computation resources. Proactive protocols propagate all topology changes throughout the network. If some node leaves the network, all other nodes receive a corresponding notification, even if the topology change does not affect the node.

Reactive routing protocols only maintain routes to destinations the device communicates with. Addresses of all devices along the towards the destination are included in the packet header. This means that source routing is independent of routing information maintained by other nodes. Reactive protocols only maintain routes to nodes of interest, which they expect to communicate with. This fact corresponds with the DCP binding concept. A potential drawback of reactive protocols is the delay for establishing the route to destination. Each DCP binding is established only once, and during this procedure the protocol would also perform a route discovery procedure, resulting in an overall delay that should be acceptable by the user. Another potential drawback of reactive protocols is an overhead in the DCP packet header that contains the route information. Reactive protocols can easily detect and avoid routing loops by simply checking that the same device is not included twice or more in the route.

DCP architecture is based on logical bindings between devices. A DCP device can only talk to devices it has established a valid binding to. After the installation procedure in which the DCP bindings are established, the DCP devices are expected to be static and not move. The size of a typical DCP network that includes the network layer and a routing mechanism is difficult to estimate, but approximately 3-5 hops should be representable for most common scenarios. DCP devices that are battery powered and movable are not expected to participate in the routing mechanism.

4.13.2 Routing in DCP IEEE 802.15.4 Networks

Source routing should be a good choice for small to medium sized static DCP networks implemented upon IEEE 802.15.4 technology with maximum 3-5

hops, and this section discusses source routing in DCP IEEE 802.15.4 networks. Note that no simulations or theoretical analysis have been performed as a part of the thesis, as the network layer investigation was not included in the thesis specification. Note that the purpose of this entire Section 4.13 is not to provide an in-depth routing protocol investigation in any sense, but rather establish a foundation for future work to be done on the subject by including a short introduction and discussion.

The route discovery procedure can be implemented by broadcasting route request messages with specified node address we are trying to locate, and maximum number of hops the route request packet can travel. All nodes should maintain and periodically update a neighbor table (all nodes within the transmission range). The packet is sent from node to node, and address of each node it travels through is added to the packet header. A node that knows the route to the node specified in route request answers with a route response message. The route response message travels back to the source by reversing the route information accumulated during the route discovery procedure. If the source node receives more than one route response, the route needs to choose the best route based on some route quality metrics. Perhaps the most simple metric is the route length measured in number of hops, making the shortest route the best one. However, more complex metrics could be used including the link congestion level, transmission delay etc.

Each route could be periodically checked for consistency by sending and receiving "hello" messages.

Each time the application sends data, the entire route information is incorporated into the network layer packet header. If the packet propagation fails along the way, the notification should be sent back to source with the address of the unavailable node, and the source should start a new route discovery procedure.

The maximum size of the IEEE 802.15.4 PHY layer packet is 127 bytes. In order to find the maximum packet size at the network layer, the IEEE 802.15.4 MAC header size must be substracted. Maximum size of the MAC header is 25 bytes (although it can vary and be smaller), according to [LRW03]. This leaves 102 bytes for the network layer header including the routing information and the DCP packet. Based on the expected small number of routing hops and small sizes of DCP packets, 102 bytes should be sufficient for many applications. A way to reduce the network packet overhead is to use the short 16-bit IEEE 802.15.4 addresses, instead of extended 64-bits addresses.

It should be noted that an introduction of a network layer represents a theoretical conflict because the maximum DCP packet size is set to 100 bytes. If the theoretical conflict is to be avoided, the network layer should never use more than 2 bytes, which is clearly impossible. One potential solution is to reconsider the DCP packets size and reduce it. Another solution is to introduce a transport layer between the network layer and the DCP layer. The transport layer would implement a fragmentation and reassembly mechanism. The layer would accept packets from DCP, and it would break these into several smaller packets, if necessary, and transfer these separately. The receiver would collect all these fragments and reassemble the original DCP packet.

Chapter 5

Mobile Phone in Home Automation

This chapter explores the possibility of using a mobile phone as a short range remote control in home automation. For example, a mobile phone could be used to monitor and control heating, lighting and other home automation scenarios.

This chapter presents a short overview of the common software platforms available on mobile phones today and explores the possibilities of creating and downloading third-party applications. The chapter also discusses the use of the short range wireless technologies available on mobile phones today, Bluetooth [SIG01b], Infrared IrDA [IRD] and IEEE 802.11 WLAN [WLA]. Distant remote control, where a mobile phone is outside the transmission range of these short range wireless technologies, is beyond the scope of the discussion.

5.1 Motivation

There are several advantages and reasons for using the mobile phone as a remote control in home automation:

- A single, universal remote control
- Comfort and convenience
- Economy, the cost of acquiring the remote control functionality is low

- Exclusivity, having something new and modern
- The communication is free of charge, and no subscription fees are required

Mobile phone could become a universal remote control, replacing many single-use remote controls and increasing comfort of the end user. Since the user already owns the mobile phone, the cost is only limited to acquiring the software application. Mobile phones offer relatively large and clear graphical displays, suitable for developing user-friendly graphical interfaces. It would be very convenient for the user to use the mobile phone as a remote control, as the user already is familiar with using the phone and its technical properties. Because the phone batteries are charged periodically, the power supply should not be an issue. The cost and complexity of transforming a mobile phone into a potentially universal remote control are relatively low as it is only the matter of downloading a new software application. Using the mobile phone as a remote control is potentially very economical.

There are several scenarios where a short-range mobile phone remote control would be useful, including the following examples:

- Lighting control
- Controlling the home theater and entertainment systems
- Opening, closing, locking and unlocking doors
- Heating, ventilation and air conditioning control
- Activating and deactivating the security systems
- Monitoring and controlling various industries processes

Figure 5.1 illustrates a scenario where a mobile phone is used to control a thermostat by transferring the desired temperature setpoint.

5.2 Software Platforms

This section presents software platforms commonly found on mobile phones today. Unlike the marked of personal computers where few systems like Microsoft Windows dominate, many operating system and platforms exist for

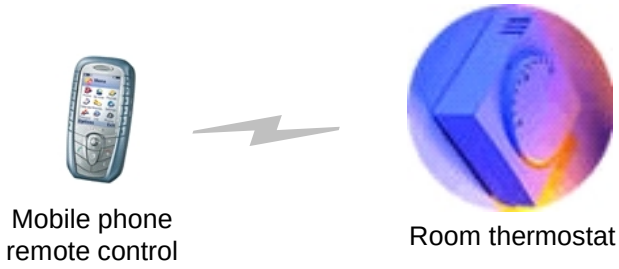

Figure 5.1: Thermostat control scenario: the mobile phone transfers the desired temperature setpoint wirelessly to the thermostat

mobile phones. For example, almost all mobile phone manufacturers have own, proprietary platforms and programming interfaces. It is therefore challenging to create interpretable applications that would run on all platforms and thus all mobile phones.

5.2.1 SymbianOS

Symbian Operating System (SymbianOS) is an advanced and open operating system for mobile phones. The platform is open in the sense that it provides a set of APIs for third party application development. Symbian [SYM] was established as an independent company in June 1998 and is owned by Nokia, Panasonic, Psion, Samsung Electronics, Siemens and Sony Ericsson. The SymbianOS programming language is Symbian C++, a variant of C++. Today, SymbianOS is implemented in more expensive professional mobile phones from various manufacturers.

5.2.2 PalmOS

Palm Operating System (PalmOS) is an operating system for handheld devices (mostly PDAs) and more powerful smartphones. PalmOS was introduced in 1996 by PalmSource Inc. [PALa], and today almost two out of every three handhelds are based on PalmOS. In addition to handheld devices and PDAs from PalmSource Inc., the PalmOS is implemented in PDAs from Sony, Handspring, IBM and several other manufacturers [PALb].

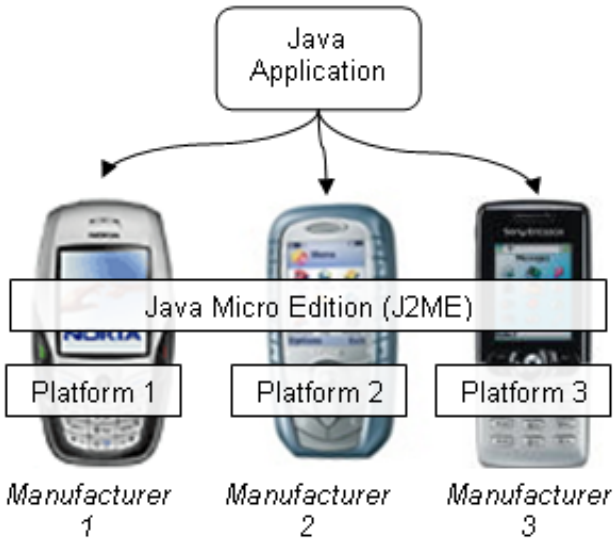

Figure 5.2: J2ME applications are platform independent. J2ME functions as a middelware layer across platforms from different manufacturers.

5.2.3 Mophun

Mophun is a platform for developing games for mobile phones. Mophun is released by Synergix Interactive [SYN]. Today, over 150 games have been developed for the Mophun platform. Mophun supports both 2D and 3D game development.

5.2.4 Java Platform 2 Micro Edition (J2ME)

Java Platform 2 Micro Edition (J2ME) [J2M] is an edition of Java 2 platform developed and optimized for handheld and embedded devices. Applications written for J2ME are platform independent, they run on all devices that implement a Java Virtual Machine (JVM). Therefore, J2ME supports the Java design philosophy "Write once, run anywhere". Today, almost all new mobile phones offer a JVM, and possibility to download and install new J2ME applications using a wireless connection. J2ME provides a standardized programming interface to Bluetooth [JSR02], but it does not provide an interface for Infrared. WLAN is supported through a Java socket interface.

5.2.5 Binary Runtime Environment for Wireless (BREW)

Binary Runtime Environment for Wireless (BREW) is a platform for wireless applications development, device configuration, application distribution, and billing. BREW was released by Qualcomm [QUA] in February 2000. Originally, BREW was restricted to work only in CDMA networks, but is now able to run in all network types. Like J2ME, BREW is also independent of the underlying platform, and it can run on many different operating systems. One of the main advantages of the BREW platform is its low memory requirements (i.e. 150 Kb). BREW supports several programming languages like C/C++, XML, Java etc.

5.2.6 Mobile Phone Platforms from Microsoft

Microsoft [MIC] has released a number of platforms targeted for handheld devices and mobile phones:

- Windows Mobile
- PocketPC
- Windows CE.NET
- SmartPhone OS

The mobile development is supported by a range of development tools, including Microsoft Visual Studio. Applications are based on the .NET Compact Framework, which uses the same programming model and very similar APIs to the .NET Framework.

5.3 Communication Technologies

This section discusses the use of the short range wireless technologies in mobile phone remote control applications. Technologies commonly implemented on mobile phones today are Bluetooth [SIG01b], Infrared IrDA [IRD] and IEEE 802.11 WLAN [WLA].

5.3.1 Bluetooth

Bluetooth [BTS] [SIG01b] offers a transmission range of 10-100 meters, and a bandwidth of 1 Mbit/s. Bluetooth communication is robust and reliable even in environments with much noise and interference. Optional security and encryption is available. However, in a home automation network Bluetooth has to be implemented into real products like thermostats, light switches, sensors etc. Some potential disadvantages of Bluetooth in this context are:

- Bluetooth allows up to 7 simultaneous connections, which might be insufficient in some scenarios
- No routing mechanisms in scatternet networks are defined by the standard.
- Bluetooth power consumption can be too high for battery powered devices where battery is not expected to be recharged periodically
- The Bluetooth inquiry (device scan) and connection procedures are time consuming

In order to avoid the Bluetooth disadvantages stated above, a possibility is to incorporate Bluetooth technology only in selected gateway products that are expected to communicate with Bluetooth remote control devices like mobile phones, but also PCs and PDAs. The scenario is illustrated in Figure 5.3, where a gateway device implements both Bluetooth and ZigBee technologies, allowing a Bluetooth based mobile phone application to indirectly communicate with ZigBee devices. In such scenarios, Bluetooth is only used for point to point connections, and the delay for setting up the connections should be acceptable.

5.3.2 IEEE 802.11 WLAN

IEEE 802.11 Wireless LAN (WLAN) [WLA] is not a very common technology on mobile phones, and only a few mobile phones today offer WLAN. WLAN is usually implemented on PDAs and PCs rather than mobile phones. WLAN is a replacement of the traditional wired network and its services, for example internet access. WLAN offers higher transmission range and bandwidth higher than Bluetooth. However, this results in a significantly higher power consumption. The WLAN protocol stack is more complex, putting

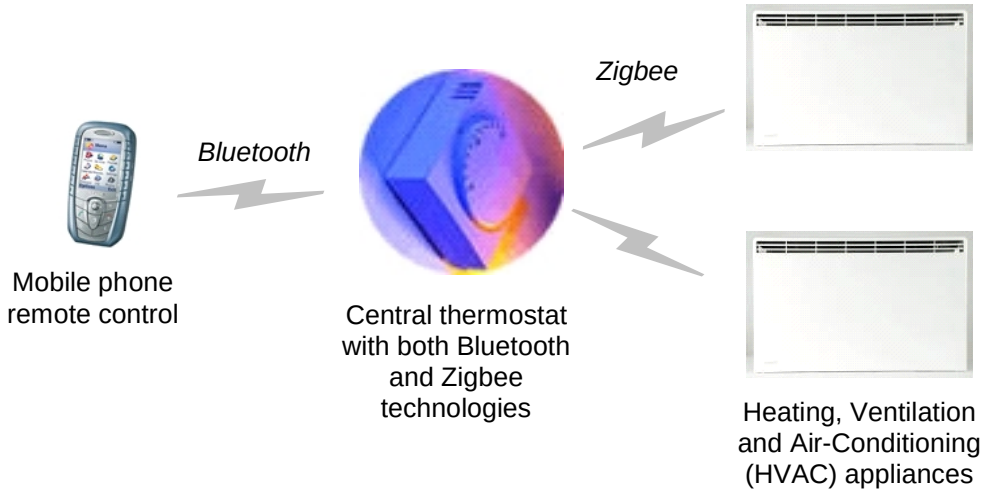

Figure 5.3: Bluetooth-ZigBee gateway scenario. The theromstat implements both Bluetooth and ZigBee acting as a gateway between Bluetooth enabled phone and ZigBee enabled appliances.

higher demands on memory, CPU and other hardware resources. Compared to Bluetooth, WLAN is more complex and expensive without giving any particular advantages in simple home automation scenarios. It is important to remember that besides having a technology available on the mobile phone, it must be embedded into a real product that the mobile phone communicates with. Therefore, cost, complexity and power consumption are important attributes. Put together, WLAN is an overkill technology for home automation, and it is unrealistic and unnecessary to implement WLAN transceivers in small, inexpensive and sometimes battery powered home appliances.

5.3.3 IrDA Infrared

Infrared communication on mobile phones follows the standards from the Infrared Data Association (IrDA) [IRD], in order to make mobile phones from different manufacturers compatible with each other. Bandwidth of up to 4 Mbit/s is sufficient for remote control applications. However, IrDA Infrared transmission range is up to one meter, meaning that the mobile phone must be very close to the appliance it communicates with. In addition,

IrDA Infrared requires line-of-sight communication. For most scenarios, these limitations are inadequate. In some contexts though, low transmission range might be seen as increased communication security, as it is almost impossible to eavesdrop.

5.4 Discussion

This section discusses the applicability of the presented software platforms and communication technologies for creating remote control applications for mobile phones.

Several mobile phone platforms and operating systems exist. Today, SymbianOS is a popular choice for expensive professional mobile phones. Because Symbian is an independent company, several manufacturers have chosen to implement SymbianOS in their products, including Nokia, Sony Ericsson, Siemens etc. SymbianOS can be used to develop remote control applications as it offers a rich set of APIs both for graphical interface development and for wireless communication. The SymbianOS programming language is C++.

PalmOS is rarely used on mobile phones, but it is a popular and common platform for PDAs. It offers a programming interface to both Bluetooth, IrDA Infrared and IEEE 802.11 WLAN. PalmOS is applicable for remote control applications for PDAs.

Mophune is a platform optimized for developing mobile games, but it can be used to create remote control applications for mobile phones. A limited number of mobile phones implement the technology.

BREW platform is independent of the underlying platform, similar to J2ME. Compared to J2ME, the popularity of the BREW platform is limited. However, this might change in the future.

Microsoft has several different platforms for mobile phones and handheld devices. The platforms can be used to implement a graphical interface for a remote control application, and they all support Bluetooth. The major problem is the limited deployment of these platforms.

Java platform 2 Micro Edition (J2ME). J2ME is platform independent, and almost all new mobile phones implement a J2ME virtual machine. The platform offers a rich set of graphical elements. J2ME is independent of the underlying platform. It has a standardized Bluetooth API (JSR-82) [JSR02]. J2ME has no standardized API support for IrDA Infrared. WLAN is supported through a Java socket interface. Another important issue that

should be discussed is the distribution of a J2ME remote control application to the mobile phone. Various alternatives exist:

- A proprietary cable between the PC and the mobile phone
- A Bluetooth or Infrared connection between the PC and the mobile phone
- A WAP connection to download the application from a HTTP server

The most common scenario will presumably be using a WAP connection to download a WAP page containing a link to the J2ME installation file. When the user clicks on the link in the WAP browser, the mobile phone automatically recognizes the file type and asks if it should install the application. The technology is called Over-The-Air (OTA) [Ort02] provisioning.

The chapter discussed three wireless technologies: Bluetooth, IrDA Infrared and IEEE 802.11 WLAN. IrDA is considered unsuitable in most scenarios because of the very short transmission range. IEEE 802.11 WLAN is in many ways similar to Bluetooth, seen in the context of the chapter. However, only few mobile phones implement WLAN, and the technology, expensive, complex and power consuming than Bluetooth. Of the three technologies considered, Bluetooth is the most suitable technology for communication between the mobile phone and home automation devices of the three technologies considered, it is implemented on many mobile phones and many of the software platforms support it. Bluetooth is supported by all of the reviewed platforms, although in different ways. Three of the platforms, SymbianOS, PalmOS and J2ME offer a full programming support for Bluetooth by offering Bluetooth specific APIs to programmers. BREW and Mophun however do not offer any Bluetooth specific APIs. These two platforms exploit the ability of Bluetooth to wirelessly emulate the standard RS232 serial port cable connections, making the Bluetooth connection appear to BREW and Mophun as a serial port cable connection between two devices.

All of the mobile platforms evaluated in this chapter can be used to implement a graphical interface for remote control applications. However, the deployment of the platforms vary considerably. J2ME is the most widely deployed platform, and is accepted as a de facto standard. Almost all new mobile phones implement a J2ME virtual machine, regardless of the price

level and manufacturer. Bluetooth is supported by J2ME through the optional JSR-82 [JSR02] API. However, the support for Bluetooth is optional, which means that each manufacturer decides whether the support shall be included into the J2ME virtual machine. Today, this the number is limited and only approximately 20 J2ME-enabled phones support JSR-82, although the number is continuously growing.

Chapter 6

Prototype Systems

Three prototype systems have been developed to illustrate the results of the thesis and demonstrate the practical application of the Device Control Protocol (DCP):

- Wireless communication between inexpensive embedded devices
- Remote control of home appliances from a mobile phone
- Remote control of home appliances from a web page

The following sections discuss present the purpose and the technical architecture of the prototype applications separately. It is important to realize that even if the applications are based on a simple lighting control scenario, the potential application area of DCP is large.

A considerable amount of time has been used to design an architecture, implement and test the prototype applications. Although the applications include a series of functional lacks and limitations, they successfully fulfill their main requirements – demonstrating the possibilities of DCP in both wired and wireless scenarios and the possibility of creating remote control applications for mobile phones. The source code can be found on the accompanying CD-ROM.

6.1 Home Automation Communication

This system demonstrates the DCP communication between low cost embedded devices in a home lighting control scenario. The system consists of one

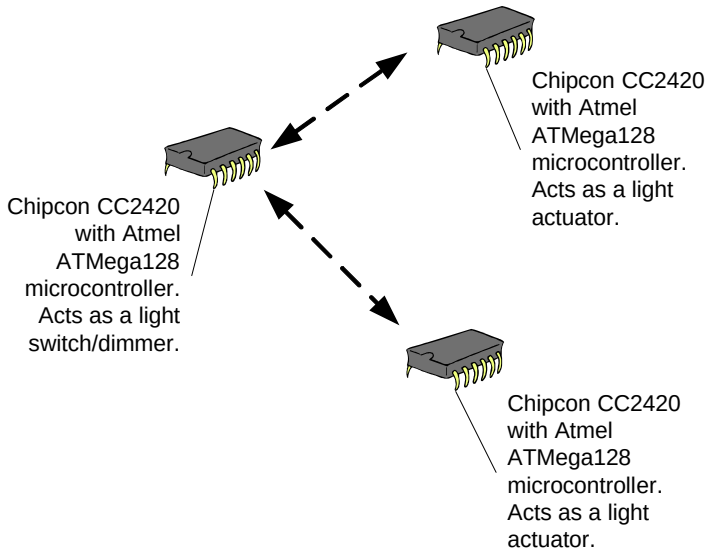

Figure 6.1: Home automation communication, architecture

light switch and one or more devices functioning as light actuators, as illustrated in Figure 6.1. A light bulb is connected to each of the light actuators to improve the demonstration effect and make it more realistic.

All communication between devices is based on the Device Control Protocol (DCP), implemented upon IEEE 802.15.4 MAC layer on a Chipcon CC2420DBK prototype board. The CC2420DBK board consists of a CC2420 2.4 GHz low-cost RF transceiver and an AVR ATmega128 microcontroller. In addition, CC2420DBK provides two buttons, four LEDs and a joystick that were used by this prototype. The ATmega128 microcontroller runs the IEEE 802.15.4 MAC software, the DCP protocol and the application software, while the Chipcon CC2420 is responsible for wireless transmission according to IEEE 802.15.4 PHY specification. CC2420DBK also allows optional 9V battery operation.

The sequence diagram in Figure 6.3 illustrates the operation mode and the communication flow of the prototype system. Moving the joystick in any direction on a light actuator makes the device establish a new wireless non-beacon IEEE 802.15.4 Personal Area Network (PAN) and become the coordinator.

Moving the joystick in any direction on a light switch makes the device

Figure 6.2: Chipcon CC2420DBK contains an 8-bit microcontroller that runs the implementation of the Device Control Protocol (DCP). The board supports the wireless IEEE 802.15.4 communication.

scan for all available PANs in the area, then tries to establish an IEEE 802.15.4 association to each of these. After a successful association, the switch creates a DCP connection and a DCP binding for the `SERVICE_SWITCH` service. The system supports incremental network expansion. At any time, moving the joystick on a light switch will start a device scan. If any new devices have been found, the switch will configure these and add them to it's binding table.

After a DCP binding has been created, pressing a button on light switch toggles the light bulbs connected to light actuator devices by sending transmitting `SETDATA_REQ` messages through the DCP binding.

Pressing the joystick centre button on a switch device removes all DCP bindings and closes all physical connections to light actuators.

The 802.15.4 technology offers 16 channels in the 2.4 GHz band, and the prototype system uses one of these. The device scan duration can be regulated by each specific implementation, as defined in [LRW03]. This prototype system uses approximately 5 seconds to scan all the 16 channels, while scanning a single channel takes less than 0.5 seconds. The transmission range of the CC2420DBK is measured to approximately 170 meters in line-of-sight. The transmission output power can be regulated in software and reduced if desired.

All communication between devices happens on a single IEEE 802.15.4 channel in a non-beaconed network in the 2.4 GHz frequency band. The switch device needs to poll the light actuator devices periodically to detect if they want to send any data to it. All three devices are Full Function Devices

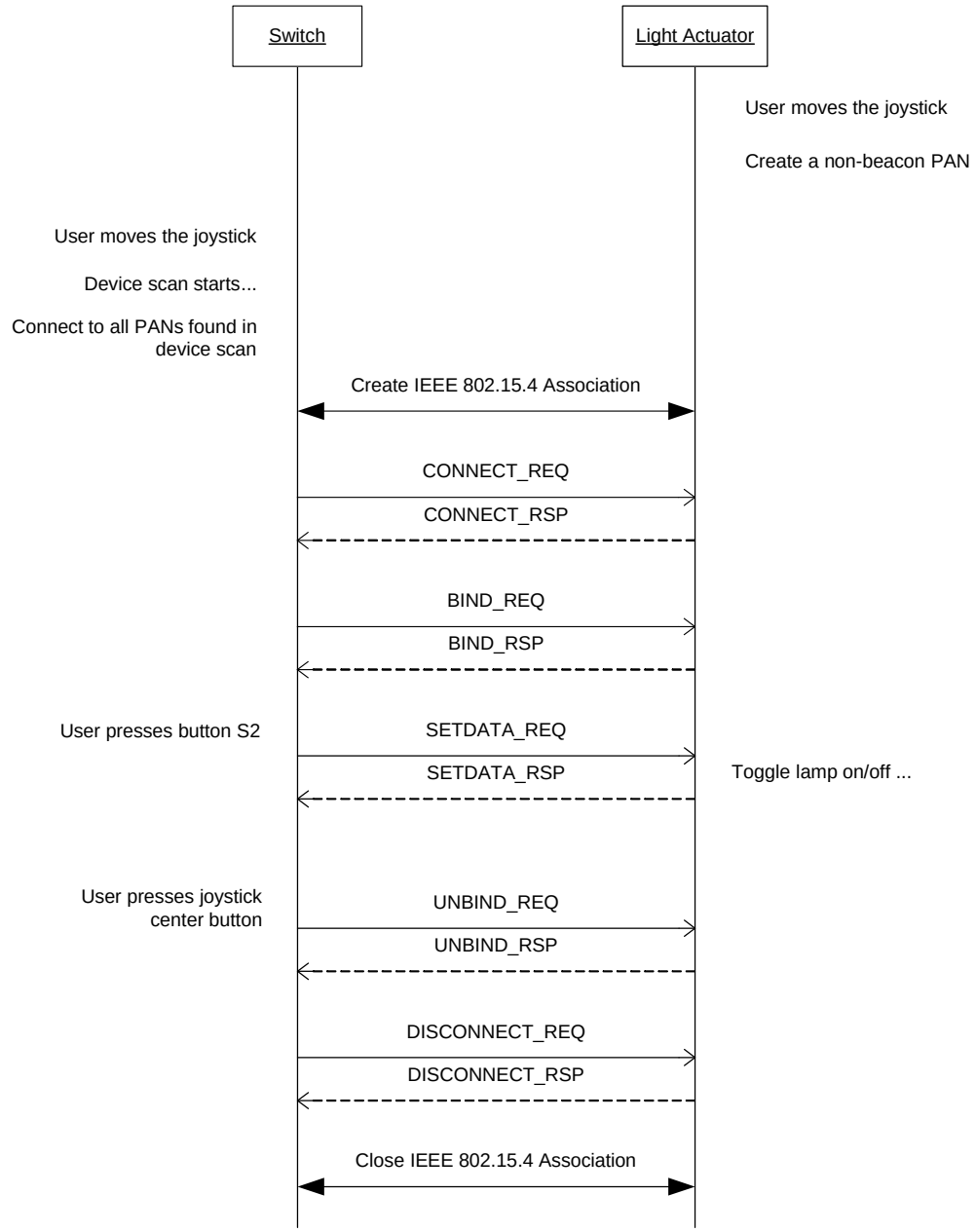

Figure 6.3: DCP communication sequence throughout the life cycle of the "Home Automation Communication" prototype system

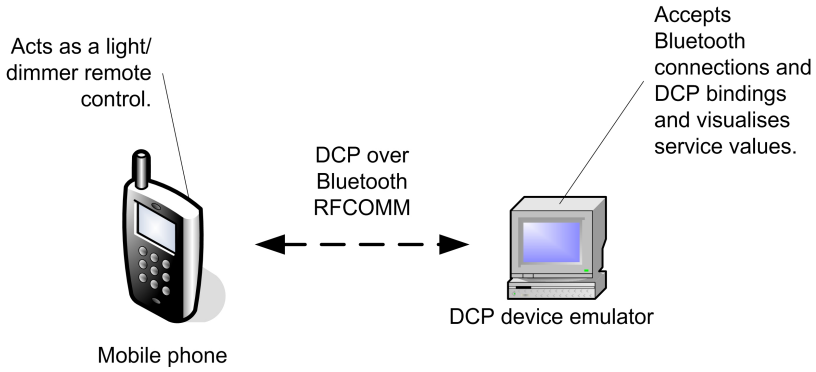

Figure 6.4: Monitoring and control from a mobile phone

(FFD). In a cost-optimized system the switch could be a Reduced Function Device (RFD).

The DCP protocol, the switch application and the light actuator application are implemented in programming language C, and compiled with the open source AVR GCC compiler version 3.3.1.

6.2 Monitoring and Control from a Mobile Phone

This system demonstrates the possibility of using a mobile phone as a short-range wireless remote control in home automation. All communication is based on the DCP protocol, implemented upon the Bluetooth RFCOMM protocol. The system architecture is illustrated in Figure 6.4, and it consists of two devices:

- A mobile phone running a Java (J2ME) application which implements a simple graphical interface for controlling a light actuator (on/off and dimming)
- A DCP device emulator running on a Linux PC and functioning as a Bluetooth enabled light actuator embedded device

The mobile phone application is developed in Java (J2ME) using the Borland JBuilder X development environment in Linux. Software that provides JSR-82 libraries was installed additionally (Series 60 MIDP Concept

SDK Beta 0.3.1), it can be downloaded from [NOK]. Being based on Java (J2ME), the mobile phone application is independent of mobile phone model and manufacturer, and is compatible with all phones with the following capabilities:

- Bluetooth wireless technology
- Java Platform 2 Micro Edition (J2ME)
- Java-Bluetooth API (JSR-82)

Approximately 20 phones support these requirements at the time of writing this, and the application was tested successfully with two of these, Nokia 6600 and Sony Ericsson P900. Although an increasingly number of mobile phones implement Bluetooth and J2ME, the support for JSR-82 is still very limited. The JSR-82 API is a relatively new specification, and not many manufacturers have implemented it yet. However, this is likely to change in the near future.

When the mobile phone application starts, it sets up a Bluetooth RFCOMM connection to the DCP device emulator, and then creates a DCP binding for the `SERVICE_SWITCH` and `SERVICE_DIMMER` services. It takes approximately 2 seconds to create a connection from the mobile phone to the DCP device emulator. This happens only once when the application starts, and it is acceptable. In addition, it takes approximately 10 seconds to perform an inquiry procedure, although this functionality was not implemented by the application. The transmission range has been measured to approximately 15 meter in line-of-sight with the available hardware (class 2 Bluetooth devices).

If the user toggles the state of the device, a `SETDATA_REQ` message is transmitted. The graphical user interface of the mobile phone application is shown in Figure 6.5.

The DCP lamp emulator is activated by pressing the "Binding" button, which makes the application listen for incoming Bluetooth RFCOMM connections and DCP binding requests. The graphical interface of the DCP lamp emulator is shown in Figure 6.6. The DCP device emulator is implemented in C and C++ using the Qt QDesigner development environment in Linux. The Linux PC is Bluetooth enabled, using a Bluetooth USB dongle hardware and the Linux Official Bluetooth stack software.

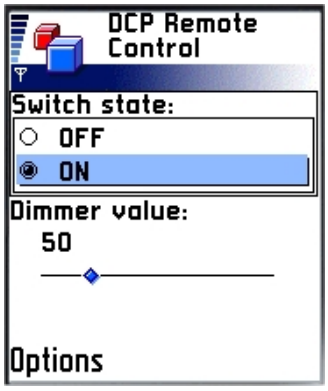

Figure 6.5: Mobile phone graphical interface, implemented in J2ME and tested both on a phone emulator and a real phone

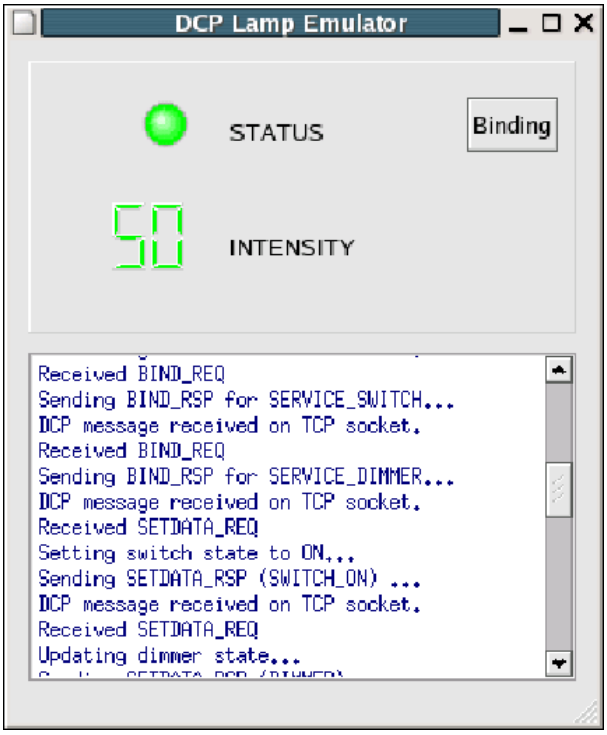

Figure 6.6: DCP lamp emulator

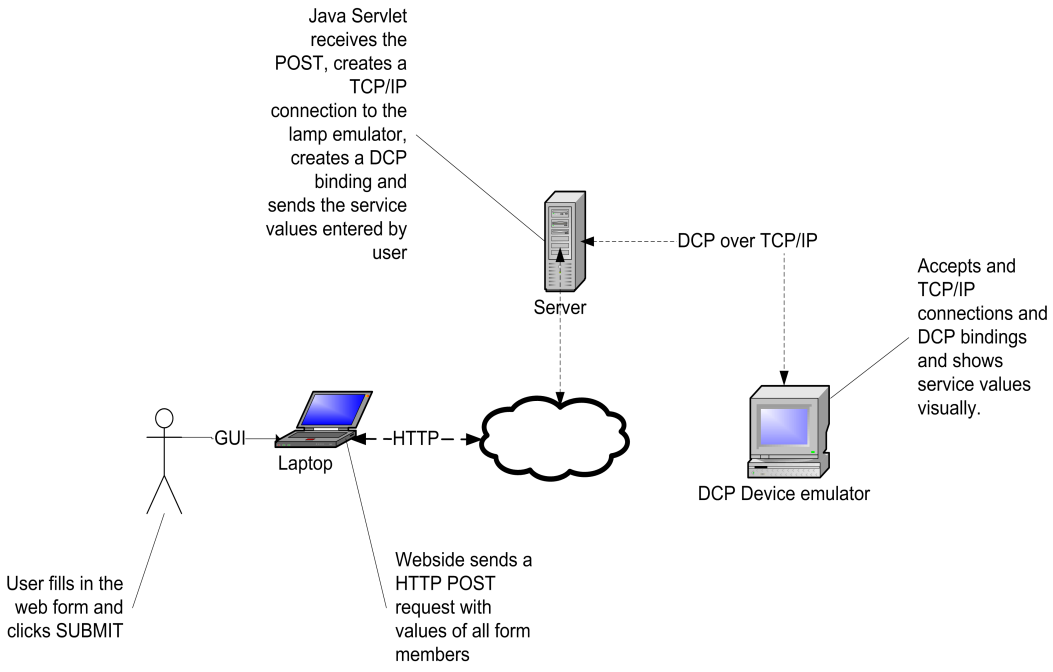

Figure 6.7: Monitoring and control from a web site

6.3 Monitoring and Control from a Web Site

This system demonstrates monitoring and remote control of home automation products from a Web site. All communication is based on the DCP protocol. The system incorporates a web page from which the used can control a lamp (emulated by a Linux application).

The system architecture is illustrated in Figure 6.7. The user fills in the information required by the web page, the IP address of the lamp emulator, status and dimmer intensity, and clicks the Submit button. User's computer creates a HTTP connection to the web server, and sends the information entered by the user in a HTTP POST message. In this prototype system, the web server runs locally on the same machine where the web page is displayed. The web server extracts the values of the information from the HTTP POST message, establishes a TCP/IP connection to the DCP device emulator and then sets up a DCP connection and a DCP binding for the `SERVICE_SWITCH` and `SERVICE_DIMMER` services.

The web interface is shown in Figure 6.8. In a full scale system, the server would be located with the provider of the remote control service. The web interface would also be improved, and instead of typing the IP address of a device, the device would be presented with a picture and a friendly name. The system would also require a logon procedure.

The application that runs on the web server is developed and implemented as a Java Servlet, although other scripting languages that offer an interface for setting up TCP connections, usually through a socket interface, could be used (e.g. ASP or PHP). This prototype system has been tested on a Jakarta Tomcat version 4.0 web server.

The DCP device emulator used in this prototype system is actually the same application described in Figure 6.6. The emulator supports both Bluetooth connections and TCP connections. The emulator is activated by pressing the "Binding" button.

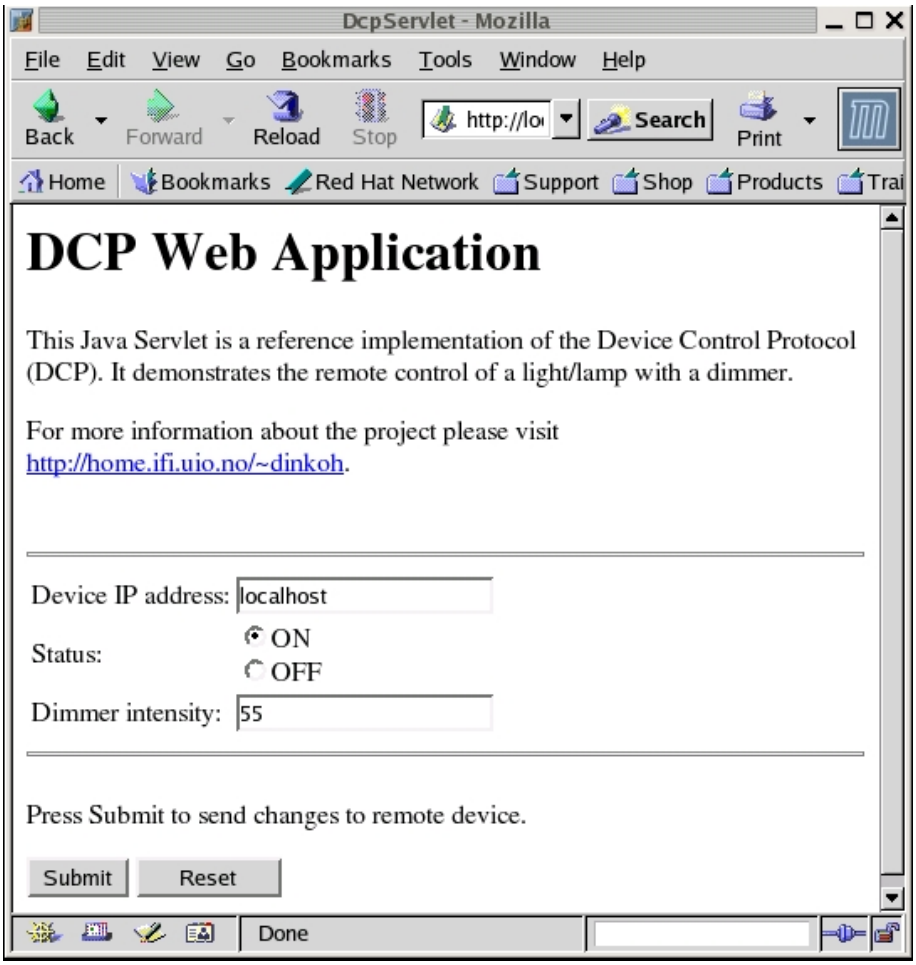

Figure 6.8: The graphical interface of the web site remote control

Chapter 7

Discussion

This chapter is a discussion of the main results of the thesis. The chapter also points out and recommends the further work to be done in succession of this thesis.

The chapter is divided into three sections, corresponding to the logical structure of thesis. Section 7.1 discusses the results of the theoretical investigation in Chapters 2 and 3, Section 7.2 discusses the Device Control Protocol (DCP) while the Section 7.3 discusses the mobile phone utilization as a short-range wireless remote control.

7.1 Theoretical Investigation

Chapter 2 briefly presents an overall review of several home automation technologies. The market is fragmented, and none of the technologies have managed to become a de facto standard in home automation yet. The market penetration is limited for most of the home automation systems that exist today. Currently, X-10 [X10a] is the most widespread technology. The deployment of the wireless systems is even more limited, compared to wired systems.

Chapter 3 presents three globally standardized short-range wireless technologies in-depth: IEEE 802.15.4 [LRW03], ZigBee [ZIG] and Bluetooth [SIG01b]. IEEE 802.15.4 and ZigBee are designed and optimized for operation in home automation scenarios and similar areas like building and industry automation. The two technologies are related, with the IEEE 802.15.4 defining the lower layers and ZigBee defining the upper layers of a protocol stack. Zig-

Bee is forecast to gain a significant growth during the next few years [Sol]. However, one needs to be careful with such predictions. A wireless technology called HomeRF [HRF], that was in many ways similar to ZigBee, was developed for a broad range of interoperable consumer devices, being backed up by over 100 companies [HRF]. Despite optimistic forecasts, HomeRF was abandoned and disbanded in January 2003.

Bluetooth was primarily developed as a cable replacement technology, not optimized for home automation. The limited number of simultaneous connections, relatively high battery consumption and time consuming device scan and connect procedures makes the technology unsuitable for some wireless home automation scenarios, although the Bluetooth Special Interest Group (SIG) [BTS] has the ambition of improving the mentioned disadvantages.

7.2 Device Control Protocol (DCP)

The thesis defines a new application-layer communication protocol suitable for use in home automation monitoring and control scenarios called the Device Control Protocol (DCP). The proposed solution fulfills the requirements given by Chipcon [CHI] in the thesis definition, offering following characteristics:

- Independency of the underlying transmission technology
- Flexibility to support a wide range of products and application areas
- Extensibility to adapt to new transmission technologies and application areas
- Simplicity to allow low cost product implementations
- Uniformity of the application program interface (API) regardless of the underlying transmission technology
- Openness and availability of the protocol specification to everyone

DCP introduces a new terminology used to ease the comprehension of important DCP mechanisms, but also ease the DCP implementation by converting the logical terminology into programming language constructs. The most important DCP terminology is summarized and discussed in following paragraphs.

Service A service is an abstraction of a device's ability to perform some specific task, as defined in Section 4.2. DCP makes it possible to define up to 2^{16} (65536) services. One half of these is reserved for standardization by DCP. The current version of the DCP specification defines only 5 standardized services (primarily because of lack of time), but an industrialized version of the DCP specification would include many more. The other half of the service quantity can be used by manufacturers to implement their own proprietary services, thus ensuring flexibility opening up for applications not covered by the standardized DCP services. The consequence of implementing proprietary service is naturally loss of interoperability with devices from other manufacturers.

Port A port is a simple integer functioning as a service multiplexer in scenarios where a device implements more than one service, as defined in Section 4.2. For example, a light switch panel with several buttons would let each button use different ports. This way, the buttons can be distinguished within the switch panel. Each device has up to 2^8 (256) ports, which means it can implement up to 256 services at the same time, which should be enough even for the most complicated DCP devices.

Binding A binding is a logical connection between two services of the same type, as defined in Section 4.2. The DCP specification limits the maximum number of bindings per port to 2^8 (256). Since the maximum number of ports is 256, the maximum number of simultaneous bindings in a device is $2^8 \cdot 2^8 = 2^{16}$ (65536). For example, a single light switch can control up to 256 light actuators. Bindings are expected to be long-lasting, and they should survive electrical power failures by being stored permanently on a device. A binding has a direction, which is considered from the viewpoint of the device initiating the binding procedure, as defined in Section 4.2. Each service is either offered to other devices (i.e. produced) or is used by some device (i.e. consumed). A binding direction is a way of determining the service producer and the consumer of each binding.

DCP offers a protocol versioning support, and distinguishes 2^8 (256) versions of the protocol, allowing the protocol to expand and evolve during several versions. A version check is mandatory in the connect procedure,

used to detect version inconsistencies between two communicating entities at an early stage.

A 16-bit DCP sequence number is included in each DCP packet, and is intended to ensure packet freshness. Sequence number of a response or an error packet always equals the sequence number of the corresponding request packet. A DCP implementation must also handle the wraparound of the sequence numbers (from 65535 to 0).

This chapter suggests a DCP API, which remains the same regardless of which underlying technology and communication protocol is used. A standardized API eases both implementation and usage of the protocol.

Section 4.13 briefly discusses the network layer routing mechanisms in DCP networks based on the IEEE 802.15.4 [LRW03] technology. Although not an in-depth investigation, Section 4.13 identifies several potential advantages of reactive protocols in preference to proactive protocols in small and static DCP networks. The section outlines a routing strategy based with properties reactive protocols. The IEEE 802.15.4 maximum packet size is 127 bytes, shared by the IEEE 802.15.4 MAC header, the network layer header, the DCP header and the DCP payload. In order to compress and reduce the network layer header size, 16 bit IEEE addresses should be used in preference to 64 bit extended addresses. If necessary, a transport layer could be implemented between the DCP layer and the network layer. The transport layer would implement a fragmentation and reassembly mechanism to break large packets into smaller ones, transmit these separately and reassemble the large packet at the receiver.

7.2.1 Further work

This subsection points out the recommended further work related to DCP.

Section 4.12 outlines a DCP bridging mechanism, which is not defined by this thesis because of time shortage. Such a mechanism would be a useful DCP service, and it would expand the DCP application area coverage. Additional DCP messages would need to defined. Addressing, device description, service discovery and binding across different transmission technologies has to be considered.

Communication security has been beyond the scope of this thesis. The further work needs to define the DCP security mechanisms, including the authentication, authorization and encryption.

Section 4.13 presents a brief discussion of the routing protocols in the

context of IEEE 802.15.4 technology and the Device Control Protocol (DCP), outlining a routing mechanism based on the characteristics of the reactive protocols. The further work in this context would both provide an in-depth theoretical investigation attempting to find the optimal routing strategy, a routing protocol simulation and a practical implementation and testing.

The DCP API function return values have not been formally specified because of the time shortage, and this is regarded as a part of the potential future work with DCP.

7.3 Mobile Phone as a Short-Range Remote Control

The thesis examines the possibility of using a mobile phone as a short-range wireless remote control in home automation, and states that several software platforms can be used to implement the graphical interface for the remote control application.

Java Platform 2 Micro Edition (J2ME) [J2M] is recommended because of the platform independency and a wide market acceptance. J2ME allows third-party applications to be downloaded and installed on the mobile phone wirelessly, using the OTA (Over-the-air) [Ort02] technology. The technology has been tested during the thesis, with the general impression of being user-friendly. The user simply visits a WAP (Wireless Application Protocol) page containing a link to the J2ME application. By clicking on the link in the WAP browser, the mobile phone automatically recognizes the application type and starts a simple installation procedure.

J2ME offers permanent storage on the mobile phone [J2M], making very easy to develop small databases stored permanently on the mobile phone. This possibility could be used to store various information about devices that are to be controlled by the mobile phone, and make the remote control application more user-friendly and reliable.

The recommended wireless technology is Bluetooth [BTS]. IEEE 802.11 WLAN [WLA] has several disadvantages compared to Bluetooth: higher battery consumption because of the increased transmission range and bandwidth, higher complexity, higher cost and much lower deployment on mobile phones. Not many phones implement IEEE 802.11 today, primarily because of the higher battery consumption. Bluetooth provides reliable, om-

nidirectional interference-resistant communication with sufficient bandwidth (1 Mbit/s at the physical layer). Although the majority of the mobile phones implement Bluetooth class 2 devices (typical transmission range of 10 meters), Bluetooth defines class 1 devices with a transmission range of up to 100 meters.

The third wireless technology considered, IrDA Infrared [IRD], is found unsuitable for most scenarios because of the very low transmission range (up to one meter) and the line-of-sight requirement. Still, it should be noted that in some scenarios where the mobile phone is used to transfer configuration parameters and similar data to a home appliance infrequently, IrDA Infrared might be considered more closely because of it's low cost, low complexity and wide deployment on mobile phones.

A demonstrator based on J2ME and Bluetooth has been developed and tested on two mobile phones, a Nokia 6600 and a Sony Ericsson P900. The application worked correctly on both phones, although some graphical components were interpreted differently on the two phones. Also, the graphical interface of the application was different on the two phones, because the colors and the appearance of the J2ME components is decided by each manufacturer. The phone application communicates through the DCP protocol with a Linux application functioning as a light actuator.

7.3.1 Further work

This subsection points out the recommended further work related to mobile phone utilization as a remote control.

Although the thesis states that it is possible to create remote control applications for mobile phones, it does not provide an in-depth study. The development of the graphical user interface is not discussed by the thesis. Communication security is also an important issue which is beyond the scope of this thesis.

In addition to short-range remote control, it would be interesting to investigate the possibilities of using such technologies as GSM (Global System for Mobile communications), GPRS (General Packet Radio Service) or SMS (Short Message Service), which would allow long-range remote control from a mobile phone. In this context it would be specially interesting to focus on the third-generation (3G) mobile wireless technologies like UMTS (Universal Mobile Telecommunications System) and the use of IPv6 (Internet Protocol version 6).

Chapter 8

Conclusion

The main part of the thesis focuses on defining a new communication protocol for home automation. I am confident to argue that the proposed solution, the Device Control Protocol (DCP), successfully meets the required characteristics. The solution focuses on simplicity, transmission-layer independency, flexibility and extensibility, and it offers a uniform protocol API. Seen in a global perspective, DCP provides a good foundation for further development and industrialization to potentially become a universal home automation communication alternative for a wide range of application areas. Various demonstrator developed during the project demonstrate the characteristics of DCP in both wired and wireless scenarios. The further work to be done should focus on communication security and a DCP bridging functionality that would make it possible to implement gateway devices forwarding DCP messages between appliances based upon different transmission technologies.

The thesis includes a brief investigation of the possibility of using a mobile phone as a short-range remote control. Mobile phone could become a convenient, economical and universal remote control. The thesis recommends the J2ME platform to host the application and the Bluetooth technology for wireless communication. However, the Bluetooth support in J2ME is optional. A limited number of mobile phones implement this support, although the number is increasing constantly. A demonstrator based on J2ME and Bluetooth was successfully implemented and tested on a mobile phone. The communication between the mobile phone and the home appliance is based on DCP.

Abbrevations

AES ..... Advanced Encryption Standard
API ..... Application Program Interface
APL ..... Application Layer
APS ..... Application Support
ASCII ..... American Standard Code for Information Interchange
ASP ..... Active Server Pages
BCI ..... BatiBUS Club International
BREW ..... Binary Runtime Environment for Wireless
CCA ..... Clear Channel Assessment
CDMA ..... Code Division Multiple Access
CEBus ..... Consumer Electronics Bus
CIC ..... CEBus Industry Council
CPU ..... Central Processing Unit
CSMA-CA ..... Carrier Sense Multiple Access with Collision Avoidance
DCP ..... Device Control Protocol
DSSS ..... Direct Sequence Spread Spectrum
EHSA ..... European Home Systems Association
EIA ..... Electronics Industry Association
EIB ..... European Installation Bus
EIBA ..... European Installation Bus Association
ERR ..... Error
FFD ..... Full Function Device
GAP ..... Generic Access Profile

GHz ..... Gigahertz
GPRS ..... General Packet Radio Service
GSM ..... Global System for Mobile communications
GTS ..... Guaranteed Time Slot
GUI ..... Graphical User Interface
HCI ..... Host Controller Interface
HTTP ..... Hypertext Transfer Protocol
HVAC ..... Heating, Ventilation and Air Conditioning
IEEE ..... Institute of Electrical and Electronics Engineers
IP ..... Internet Protocol
IPv6 ..... Internet Protocol version 6
IR ..... Infrared
IrDA ..... Infrared Data Association
ISM ..... Industrial, Scientific and Medical
J2ME ..... Java Platform 2 Micro Edition
JSR ..... Java Specification Request
JVM ..... Java Virtual Machine
Kb ..... Kilobyte
Kbit/s ..... Kilobit per second
L2CAP ..... Logical Link Control and Adaptation Protocol
LAN ..... Local Area Network
LC ..... Link Controller
LCP ..... Link Control Protocol
LM ..... Link Manager
LQI ..... Link Quality Indication
LSB ..... Least Significant Bit
MAC ..... Media Access Control
MANET ..... Mobile Ad hoc Networking
Mb ..... Megabyte
Mbit/s ..... Megabit per second
MHz ..... Megahertz

MIDP ..... Mobile Information Device Profile
MTU ..... Maximum Transmission Unit
mW ..... Miliwatt
NWK ..... Network
OBEX ..... Object Exchange
OS ..... Operating System
OSI ..... Open Systems Interconnect
OTA ..... Over The Air
PAN ..... Personal Area Network
PC ..... Personal Computer
PDA ..... Personal Digital Assistant
PHP ..... PHP Hypertext Preprocessor
PHY ..... Physical
PLC ..... Power Line Communication
PSTN ..... Public Switched Telephone Network
QoS ..... Quality of Service
REQ ..... Request
RF ..... Radio Frequency
RFCOMM ..... Radio Frequency Communications Protocol
RFD ..... Reduced Function Device
RF-IC ..... Radio Frequency Integrated Circuit
RSP ..... Response
SDK ..... Software Development Kit
SDP ..... Service Discovery Protocol
SIG ..... Special Interest Group
SMS ..... Short Message Service
TCP ..... Transmission Control Protocol
TCS ..... Telephony Control Specification
TDD ..... Time Division Duplex
TP ..... Twisted Pair
UIO ..... University of Oslo

UMTS ..... Universal Mobile Telecommunications System
UNIK ..... University Graduate Center at Kjeller
USB ..... Universal Serial Bus
WAP ..... Wireless Application Protocol
WLAN ..... Wireless Local Area Network
XAP ..... Extensible Automation Protocol
XML ..... Extensible Markup Language
ZDO ..... ZigBee Device Object

Bibliography

[Ada03] John Adams. What you should know about the ZigBee Alliance, 2003. Available online (June 2004): http://www.zigbee.com/resources.

[AH03] Ranjith Antony and Bruce Hopkins. *Bluetooth for Java*. APress, first edition, 2003.

[BCI] BatiBUS Club International (BCI). Official homepage: http://www.batibus.com.

[BS02] Jennifer Bray and Charles F. Sturman. *Connect Without Cables*. Prentice Hall, Upper Saddle River, New Jersey, second edition, 2002.

[BTS] The Bluetooth Special Interest Group (SIG). Official homepage: http://www.bluetooth.com.

[Cal04] Edgar H. Callaway. *Wireless Sensor Networks*. Aurebach Publications, first edition, 2004.

[CEB] Consumer Electronics Bus (CEBus). Official homepage: http://www.cebus.com.

[CHI] Chipcon. Official homepage: http://www.chipcon.com.

[ECB02] Lance Hester Jose A. Gutierrez Marco Naeve Bob Heile Ed Callaway, Paul Gorday and Venkat Bah. Home Networking with IEEE 802.15.4: A Developing Standard for Low-Rate Wireless Personal Area Networks. Technical report, IEEE Communications Magazine, Aug 2002.

[ECH] Echelon Corporation. Official homepage: http://www.echelon.com.

[EHS] European Home Systems Association (EHSA). Official homepage: http://www.ehsa.com.

[EIB] European Installation Bus Association (EIBA). Official homepage: http://www.eiba.com.

[HG03] Ivan Howitt and Jose A. Gutierrez. IEEE 802.15.4 Low Rate Wireless Personal Area Network Coexistence Issues. Technical report, IEEE, 2003.

[HK03] Ritsuko Kanazawa Hiromichi Ito Hiroshi Kanma, Naboru Wakabayashi. Home Appliance Control System over Bluetooth with a Cellular Phone, 2003.

[HRF] HomeRF Resource Center. Available online (June 2004): http://www.palowireless.com/homerf.

[IET] Mobile Ad Hoc Networks (MANET) Working Group. Available at http://www.ietf.org/html.charters/manet-charter.html.

[INT] Introduction to LonWorks System. Available online (June 2004): http://echelon.com/support/documentation/Manuals/078-0183-01A.pdf.

[IRD] Infrared Data Association (IrDA). Official homepage: http://wwww.irda.org.

[J2M] Java Micro Edition (J2ME). Official homepage: http://java.sun.com/j2me/index.jsp.

[JGB03] Edgar Callaway Jose Gutierrez and Raymond Barrett. *Low-Rate Wireless Personal Area Networks: Enabling Wireless Sensors with IEEE 802.15.4*. IEEE Press, first edition, 2003.

[JSR02] Java APIs for Bluetooth Wireless Technology (JSR-82) Specification Version 1.0a. Technical report, Java Community Process (JCP), 2002. Official online at: http://jcp.org/aboutJava/communityprocess/final/jsr082/index.html.

[KD] Charles D. Knutson and Glade Diviney. Infrared Data Communications with IrDA. Technical report, Infrared Data Association (IrDA). Available online (June 2004): http://www.irda.org.

[KNX] Konnex Association. Official homepage: http://www.konnex.org.

[KT02] K.N. Wang K.K. Tan, S.Y. Soh. Development of an Internet Home Control System, 2002.

[LRW02] Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). Technical report, Institute of Electrical and Electronics Engineers (IEEE), 2002.

[LRW03] Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). Technical report, Institute of Electrical and Electronics Engineers (IEEE), 2003.

[LWP] EBV Elektronik, LonWorks Area. Available online (June 2004): http://www.ebv.com/prodserv/lonworks/lonworks.phtml.

[MB00] Miller and Bisdikian. *Bluetooth Revealed*. Prentice Hall, Upper Saddle River, New Jersey, 2000.

[Met01] Dave Methvin. Exploring X-10 technology, Sep 2001. Available online (June 2004): http://www.connectedhomemag.com/homecontrols/articles/index.cfm?articleid=22668.

[MIC] Microsoft. Official homepage: http://www.microsoft.com.

[Mis99] Padmini Misra. Routing Protocols for Ad Hoc Mobile Wireless Networks. Technical report, Ohio State University, 1999. Available online (June 2004): http://www.cis.ohio-state.edu/~jain/cis788-99/adhoc-routing/index.html.

[NOK] Forum Nokia. Official homepage: http://www.forum.nokia.com.

[Ort02] Enrique Ortiz. Introduction to OTA Application Provisioning. Technical report, Sun, Nov 2002. Available online (June 2004): http://developers.sun.com/techtopics/mobility/midp/articles/ota/.

[PALa] PalmSource Inc. Official homepage: http://www.palmsource.com.

[PALb] PalmOS Devices. Available online (June 2004): http://chris.mckinney.net/palm/default.html.

[PJMK] David W. Suvak Patrick J. Megowan and Charles D. Knutson. IrDA Infrared Communications: An Overview. Technical report, Infrared Data Association (IrDA). Available online (June 2004): http://www.irda.org.

[QUA] Qualcomm. Official homepage: http://www.qualcomm.com.

[Roy99] Elizabeth M. Royer. A review of current routing protocols for ad hoc mobile wireless networks. Technical report, IEEE Personal Communications, 1999.

[SHA] Smart House article. Available online (June 2004): http://www.agentland.com/pages/learn/bots5.html.

[SHN02] Smart Home Networks - The Fight for Control, 2002. Available online (June 2004): http://www.igigroup.com/st/pages/bringlightv3.html.

[SIG01a] The Bluetooth SIG. The Bluetooth Profiles, 2001. Available online at https://www.bluetooth.org/spec/.

[SIG01b] The Bluetooth SIG. The Bluetooth Specification version 1.1. Technical report, The Bluetooth SIG, 2001. Available online at https://www.bluetooth.org/spec/.

[SIG03] The Bluetooth SIG. The Bluetooth Specification version 1.2. Technical report, The Bluetooth SIG, 2003. Available online at https://www.bluetooth.org/spec/.

[Sol] West Technology Research Solutions. ZigBee Market Report and Analysis. Available online (June 2004): http://www.westtechresearch.com/zigbee.htm.

[SYM] Symbian. Official homepage: http://www.symbian.com.

[SYN] Synergix Interactive. Official homepage: http://www.synergenix.se.

[Tie00] F. Tiersch. *LonWorks Technology : An Introduction*. Desotron Verlagsgesellschaft, 2000.

[TMSH00] Richard C. Braley Thomas M. Siep, Ian C. Gifford and Robert F. Heile. Paving the Way for Personal Area Network Standards: An Overview of the IEEE 802.15 Working Group for Wireless Personal Area Networks. Technical report, IEEE Personal Communications, Feb 2000.

[UIO] University of Oslo, Faculty of Mathematics and Natural Sciences, Department of Informatics. Official homepage: http://www.ifi.uio.no.

[UNI] University Graduate Center at Kjeller. Official homepage: http://www.unik.no.

[WLA] IEEE Working Group for WLAN Standards. Official homepage: http://grouper.ieee.org/groups/802/11.

[WPAa] IEEE 802.15 WPAN Task Group 1. Official homepage: http://www.ieee802.org/15/pub/tg1.html.

[WPAb] IEEE 802.15 WPAN Task Group 5. Official homepage: http://www.ieee802.org/15/pub/tg5.html.

[X10a] X-10. Official homepage: http://www.x10.com.

[X10b] Common X-10 Problems. Available online (June 2004): http://www.x10ideas.com/articles/displayx10article.asp?articleid=9.

[XAP] XAP home automation protocol. Available online (June 2004): http://www.xapautomation.org/index.php.

[ZEN] Zensys. Official homepage: http://www.zen-sys.com.

[ZIG] ZigBee Alliance. Official homepage: http://wwww.zigbee.com.

Appendix A

Contents of the Accompanying CD-ROM

- Report
  - PDF
  - DVI
  - Latex
- Project Website
- Source Code
  - IEEE 802.15.4 Communication
  - Mobile Phone Control
  - Web Control
  - DCP Device Emulator
- Device Control Protocol (DCP)
  - PDF
  - Microsoft Word