Lappeenranta University of Technology
Department of Information Technology
Master's Thesis

Reusable Bluetooth Networking Component for Symbian Operating System

The subject has been approved by the council of the Department of Information Technology on April 10, 2002

Supervisor: Professor Jari Porras
Instructor: M.Sc. Kimmo Hoikka

Lappeenranta, September 1, 2002

Samuli Forsman
Kaivosuonkatu 6 A 1
FIN - 53850 Lappeenranta
Finland
+358 40 709 2521
samuli.forsman@iki.fi

ABSTRACT

Lappeenranta University of Technology
Department of Information Technology
Samuli Forsman

Reusable Bluetooth Networking Component for Symbian Operating System

Master's Thesis
2002
84 pages, 31 figures, 8 tables and 3 appendices
Supervisor: Professor Jari Porras
Keywords: software reuse, software components, software design, Bluetooth, Symbian

This thesis presents different aspects of software reuse, introduces Symbian OS, which is an operating system for wireless information devices, and presents the essentials of wireless Bluetooth technology. The practical part of work was to design and implement a Symbian OS based reusable software component for Bluetooth networking.

The benefits of software reuse are very substantial. Reusable software components improve the quality and performance of software. Development-cycle of software products can be shortened significantly and the overall costs of development can be lowered when a reuse program has been efficiently adapted to software development. Nevertheless, several obstacles complicate successful software reuse. Lack of resources, training and attitudes are just mentioned as few examples.

Bluetooth technology is currently maturing; more Bluetooth enabled devices have entered the market during last two years and the need for BT applications is clearly increasing. Therefore, developing reusable Bluetooth component for Symbian OS makes sense. Component contains the basic functionality needed for BT communications between two devices.

TIIVISTELMÄ

Lappeenrannan teknillinen korkeakoulu
Tietotekniikan osasto
Samuli Forsman

Uudelleenkäytettävä Bluetooth-ohjelmistokomponentti Symbian-käyttöjärjestelmälle

Diplomityö
2002
84 sivua, 31 kuvaa, 8 taulukkoa ja 3 liitettä
Tarkastaja: Professori Jari Porras
Hakusanat: ohjelmistojen uudelleenkäyttö, ohjelmistokomponentit, ohjelmistosuunnittelu, Bluetooth, Symbian

Tässä diplomityössä käsitellään eri näkökulmia ohjelmistojen uudelleenkäyttöön sekä esitellään perustiedot langattomiin laitteisiin käytettävästä Symbian-käyttöjärjestelmästä ja langattomasta Bluetooth-teknologiasta. Työn käytännön osuudessa suunniteltiin ja toteutettiin uudelleenkäytettävä Bluetooth-ohjelmistokomponentti Symbian-käyttöjärjestelmälle.

Ohjelmistojen uudelleenkäytön edut ovat erittäin selkeitä. Uudelleenkäytettävät ohjelmistokomponentit parantavat ohjelmiston laatua ja suorituskykyä. Ohjelmistotuotteiden tuotekehityssykliä voidaan lyhentää merkittävästi ja kehitystyön kokonaiskustannuksia voidaan alentaa tehokkaalla uudelleenkäyttöohjelmalla. Kuitenkin uudelleenkäytöllä on myös esteitä, esimerkkeinä näistä ovat mm. resurssien puute, koulutus sekä uudelleenkäytön vastaiset asenteet.

Bluetooth-teknologia on kypsynyt viimeisen kahden vuoden aikana, kun markkinoille on tullut yhä enemmän Bluetooth-laitteita ja niitä käyttäviä sovelluksia. Kehitetty komponentti tarjoaa perustoiminnallisuudet Bluetooth-yhteyksien muodostamiselle ja datan siirtämiselle laitteiden välillä.

PREFACE

My “career” with computers started on Christmas 1988 when my parents bought me Commodore 64, a sensational and advanced home computer of that time. At first, it was just for playing games but later on, also programming started to interested me. In the beginning of 1990’s I bought my own PC and during high school I started to think that maybe I could do my daily work with computers on some day... In 1995, on the last year of high school, I decided to study Information technology in the Lappeenranta University of Technology. Last five years have really been the best years of my life so far, but now it is (finally) time to graduate ☺

I would like to warmly thank Kimmo Hoikka for instructing this thesis. He gave me several good ideas and improvement proposals during the whole thesis process. Kimmo is also a tremendous expert on software engineering and his instructions have helped many employees of Digia several times. I thank also Jari Hakulinen for his comments to this thesis.

Jari Porras, the supervisor of this thesis, is definitely worth of mentioning. Jari gave me good guidance for making thesis. He has also the ability to teach students, which is a very fine characteristic for a professor of university.

Finally, the biggest thanks go to my parents; they have always encouraged and supported me to study.

“*If you are indecisive, you’ll get nothing done.”*

- *Chinese proverb*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

Abbreviation	Description
AFE	Application Family Engineering
API	Application Programming Interface
ASE	Application System Engineering
BT	Bluetooth
C32	Symbian OS Comms server
CDMA	Code Division Multiple Access
CPU	Central Processing Unit
CSE	Component System Engineering
CSY	Communications server module
DBMS	Database Management System
DFRD	Design Family Reference Design
DLL	Dynamic Link Library
DSP	Digia Software Process
EDGE	Enhanced Data rates for GSM Environment
ESOCK	Symbian OS Socket server
ETEL	Symbian OS Telephony server
FHSS	Frequency Hopping Spread Spectrum
GFSK	Gaussian Frequency Shift Keying
GHz	Giga Hertz
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
GT	Symbian OS Generic Technology
GUI	Graphical User Interface
HCI	Host Controller Interface
HTTP	Hypertext Transfer Protocol
HW	Hardware

IMAP	Internet Mail Access Protocol
IPC	Inter-Process Communication
IPv4 / IPv6	Internet Protocol version 4 / 6
IrDA	Infrared Data Association
ITC	Inter-Thread Communication
J2ME	Java version 2 Micro Edition
L2CAP	Bluetooth's Logical Link Control and Adaptation Protocol
LAF	Look-And-Feel
LDD	Logical Device Driver
LMP	Bluetooth's Link Manager Protocol
LOC	Lines of Code
MAGIC	Modular Architecture for Generic Interface Components
MIDP	Java Mobile Information Device Profile
MMS	Multimedia Message Service
MTM	Message Type Module
OMA	Open Mobile Alliance
OOM	Out-of-memory
OS	Operating System
P2P	Peer-To-Peer
PDA	Personal Digital Assistant
PDD	Physical Device Driver
PIM	Personal Information Management
POP3	Post Office Protocol version 3
PRT	Symbian OS protocol module
RFCOMM	Bluetooth's Serial line emulation protocol
RS232	Standard for serial communications
SCV	Scope, commonality and variability analysis
SDK	Software Development Kit
SDP	Bluetooth's Service Discovery Protocol

SIM	Subscriber Identification Module
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SOS	Symbian Operating System
SW	Software
SyncML	Technology for synchronizing the user data of wireless information device
TCP/IP	Transmission Control Protocol / Internet Protocol
TSY	Symbian OS Telephony server module
UI	User Interface
UIQ	User interface for Symbian OS version 7.0.
UML	Unified Modelling Language
USB	Universal Serial Bus
WAP	Wireless Application Protocol

1. Introduction

1.1. Background information

Software systems have grown enormously during last decades and they have spread all over the modern world to our daily lives. Business and market drivers set demands for software projects and fulfilling them require considerable development efforts. Therefore, the significance of reuse has grown remarkably in software industry. Software reuse has been efficiently enabled by design principles, patterns and component based systems. They have also had a great influence on specifying, designing, implementing and testing modern object-oriented software systems.

1.2. Goals of the thesis

This thesis work was done for Digia Inc. during spring and summer 2002. Digia is a Finnish mobile software company established on 1997 and today Digia produces technologies and software products for Symbian OS mobile phones. Currently Digia is moving it's own product development more towards component based software development and this is one reason for the topic of this thesis. Therefore, the theoretical part of the thesis handles software reuse related issues.

The other main topic of thesis is Bluetooth technology. The practical part of work was to design and implement a reusable software component for Bluetooth networking in Symbian OS environment. Symbian OS provides a socket based interface for all supported communications bearers but there is also a clear need for an easier programming interface. Usage of Bluetooth protocol implementation isn't always so straightforward, by providing the application developer an easier Bluetooth component, we can fasten the development and make it significantly easier to enable Bluetooth and networking in applications.

1.3. Structure of the thesis

The structure of this thesis is following: chapter 2 gives a brief introduction to Symbian Operating System and also more detailed look for communication subsystem is presented. Chapters 3 and 4 handle issues related to software reuse, its technical and nontechnical aspects, component based software engineering and processes.

Chapter 5 includes basic information about wireless Bluetooth technology. Features of Bluetooth protocol stack, BT profiles and Symbian OS Bluetooth interfaces are presented in that chapter. Chapter 6 concentrates on the practical part of thesis; architecture, design and implementation issues of reusable Bluetooth software component are handled there. Chapter 7 is the last chapter of this thesis and it concludes the lessons learnt during thesis.

Unified Modelling Language (UML) is used as a notation language of component architecture, class diagrams etc. in this thesis. UML notation as used in Digia's software projects and defined in Digia Software Process / 1 / is presented in appendix 1.

2. Symbian Operating System

2.1. Introduction

Symbian is a British software company that develops Symbian Operating System (formerly known as EPOC) for Wireless Information Devices, such as smartphones, communicators and PDA's. Symbian was established in 1998 and its main owners are Ericsson, Nokia, Sony-Ericsson, Motorola, Psion, Siemens and Matsushita (Figure 1, / 24 /). Company has offices in UK, Japan, Sweden and USA.

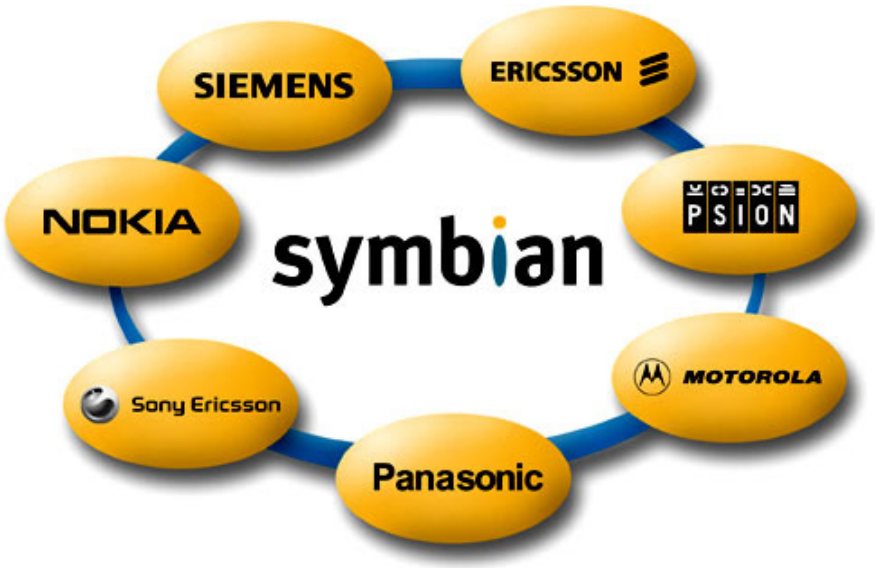


Figure 1: Owners of Symbian

Symbian OS is an advanced 32-bit operating system designed specially for demanding requirements of smartphones and other wireless information devices. Currently available Symbian OS mobile phones (Nokia 9210 communicator and Nokia 7650 smartphone) are based on version 6.X of Symbian OS. Sony-Ericsson will participate also to Symbian OS smartphone markets with its P800 smartphone during autumn 2002. P800 is based on the newest version of operating system, Symbian OS version 7.0.

The main characteristics that differenties Symbian OS from other major competitors (Microsoft's Windows CE and PalmOS) are open application environment, open standards and interoperability, multitasking, flexible user interfaces and robustness / 2 / .

Open standards and environments are significant changes in commercial mobile phone industry. For instance, Nokia started Open Mobile Alliance (OMA) in the end of year 2001 to form an open, interoperable environment for mobile devices, software and services in the future / 3 / . Several players in mobile market have participated in OMA and Symbian OS has a significant role in it as the main software platform for forthcoming mobile services and applications.

From technical point of view, robustness, real multitasking and flexible user interfaces are important characteristics of Symbian Operating System. These features are essential in developing software for demanding environments such as mobile phones and other wireless devices truly are. Applications in these devices may be up-and-running for several weeks (even months) and during this time they must always be in stable state and the user-data must not lost in any circumstance.

Flexible user interfaces are a major asset in Symbian OS. Symbian OS can be divided to two main parts (Figure 2); Generic Technology (GT) and User interface (UI). Generic technology forms the base for operating system features and User interface is just the surface for applications. User interface contains the look-and-feel (=how application data is shown to user and how application and its data are used) for applications and methods for inputting user data. Each device manufacturer implements its own drivers for their proprietary hardware.

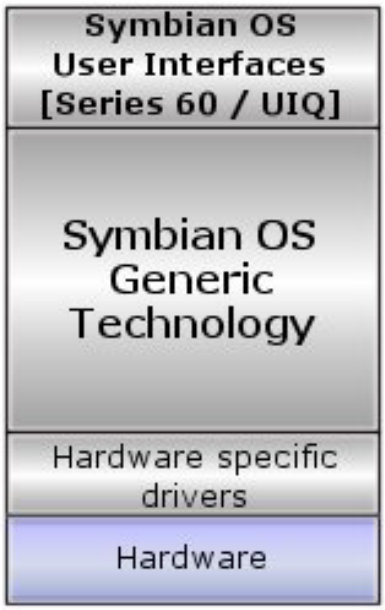


Figure 2: Symbian OS GT and User Interface

2.2. Symbian OS Generic Technology

Symbian OS Generic Technology offers the basic features and services for whole operating system. In Figure 3 is presented the main building blocks of GT in Symbian OS version 7.0, a brief introduction to each one is in following table / 2 / .

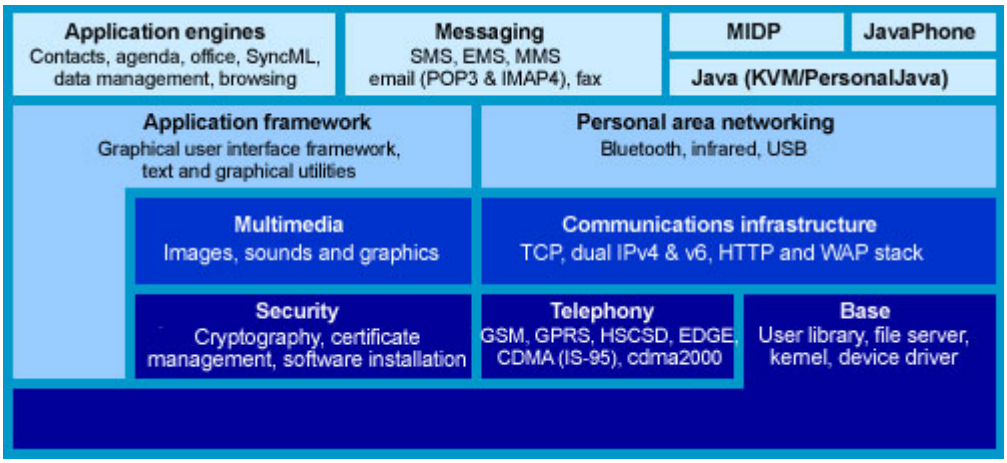


Figure 3: Symbian OS architecture

GT feature:	Description:
Application Engines	Several application engines for common Personal Information Management (PIM) applications. Symbian OS version 7.0 contains engines i.e. for agenda, contacts and office applications. Mobile phone manufacturers or individual software developers can utilize these engines in their PIM applications with their own individual user interfaces. SyncML engine can be used for synchronizing user data and Web engine contains basic features for developing web browsers.
Messaging	Symbian OS Messaging framework is used for messaging features needed in modern smartphones. Multimedia Messages (MMS) and short messages (SMS) are examples of framework services. Each messaging feature is implemented as Message Type Module, which is further presented on chapter 2.4.8.
Java	Symbian OS supports two different versions of J2ME Java: Personal Java and J2ME MIDP. The first one is intended for Communicator-type devices (e.g. Nokia 9210) with bigger resources (screen, CPU, memory) than ordinary smartphones. J2ME MIDP is the choice for smartphones or other very memory-constrained devices and it offers execution environment for midlets, which are Java applications for J2ME MIDP enabled devices.
Application Framework	The main subcomponents of Symbian OS Application Framework are: • Application execution in separate processes • Graphical User Interface framework for application UI implementations, GUI framework also makes porting application from one Symbian OS User Interface environment to another easier • Support for internationalisation of applications etc.
Personal Area Networking	Symbian OS features for local networking. Technologies for these purposes are Bluetooth, infrared and version 7.0 of operating system offers also support for USB link cables.

Multimedia	Subsystem for multimedia features. Media server provides services for audio and graphics related functionality.
Communications infrastructure	Subsystem for all data communications related features. The main components are: • Communications database for system-wide information storage • Servers for socket and serial communications • IP networking support (IPv4 and IPv6) • HTTP and WAP stacks
Security	Symbian OS security framework includes cryptographic, certificate management and secure software installation modules.
Telephony	The subsystem for features needed in mobile phones. Symbian OS v7.0 supports i.e. GSM, GPRS, EDGE and CDMA technologies.
Base	Subsystem that provides programming framework for all other components of operating system. Kernel and User libraries, device drivers and file server are the main building blocks of the base of Symbian OS.

Table 1: Building blocks of Symbian OS GT

2.3. Symbian OS user interfaces

Symbian Operating System's application framework has been designed so that only minor changes should be needed when the user interface of application or even device itself changes. The major application logic and several UI components are the same in different versions; only the upper-most layer of user interface is replaced with UI component library of particular reference design.

Until version 5 of Symbian's operating system (EPOC Release 5, ER5), the only user interface was called EIKON GUI, which was optimised for Psion's PDA devices (large screen and full keyboard). Symbian released new design family reference designs (DFRD)

in version 6.0. These reference designs were Quartz for wireless information devices with touch screen (palmtop devices) and Crystal for devices with large screen and full keyboard. Currently there are no Quartz based devices in consumer markets. Nokia Communicator 9200 series utilizes Crystal reference design on its user interfaces; chapter 2.3.3 gives more information about this.

Symbian released new user interface in OS v7.0, called UIQ. During year 2001 Symbian itself concentrated more on GT features so the UIQ user interface has been developed in co-operation of Symbian and Sony-Ericsson. Another current user interface framework is Nokia's Series 60 Platform. Both of these are presented briefly in next chapters.

2.3.1. Mobile phones with a numeric keypad - Series 60 Platform

Nokia announced in autumn 2001 that it is going to licence smartphone platform, called Series 60 Platform, also for other mobile phone manufacturers. Series 60 is based on Symbian OS v6.1 Generic Technology and the user interface called AVKON, and it is specially intended for smartphones (small devices with only numeric keypad, see Figure 4). Series 60 Platform contains also some basic applications (e.g. Personal Information Management, Synchronization, GSM telephony applications, etc.) needed in all mobile phones. / 4 / / 31 /

The main reason for Nokia to licence the Series 60 Platform for its competitors is to ensure that "the new brave mobile world" will be developed using common and open standards and to ensure the interoperability between different devices and services. In addition, a fear of Microsoft becoming too dominant player in mobile market has been obvious in Series 60 related business activities. Currently, many major mobile phone manufacturers have showed their interest to licence Series 60 and to make smartphone products based on it (at the moment i.e. Siemens and Matsushita (=Panasonic) have already licensed the platform). Nokia's first Series 60 based mobile phone is Nokia 7650 camera phone, which entered the market on summer 2002.

Figure 4: Series 60 Platform, user interface and application platform for smartphones

2.3.2. Mobile phones with touch screens – UIQ

Symbian and Sony-Ericsson in co-operation developed the user interface for the newest version of operating system. It is called UIQ, and as the Series 60, it provides some basic personal information management and messaging applications / 5 / . Touch screens are often easier and faster to use, but on the other hand, they are more expensive to manufacture than ordinary screens. Sony-Ericsson's first UIQ product, Sony-Ericsson P800 (Figure 5), will be launched to customer markets during autumn 2002.

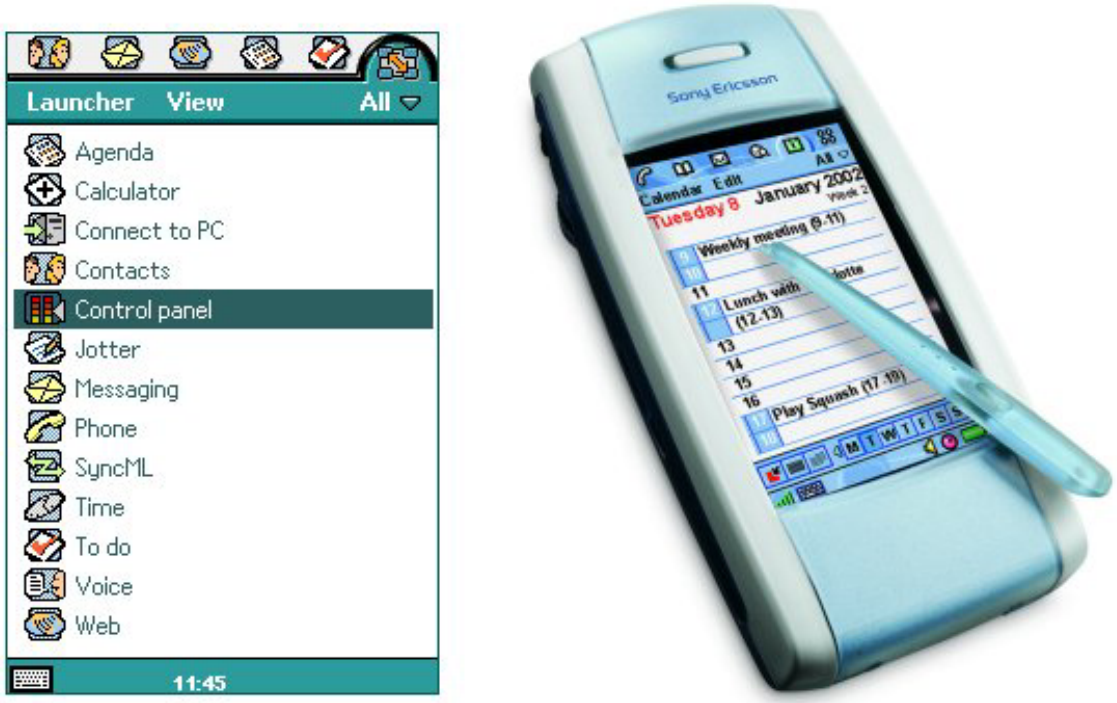


Figure 5: UIQ user interface for devices with touch screen

2.3.3. Mobile phones with full keyboard and large screen - Crystal reference design

Nokia's Communicator device family is currently the only device that utilizes Crystal reference design (also known as Nokia Series 80 Platform) for Symbian OS user interface. The reason for this is that devices with large screens and full keyboard are very expensive to manufacture, this is also a natural reason for quite high prices in consumer markets. However, large screen offers possibilities to develop applications that would be completely unusable with smaller devices (e.g. web browsers).

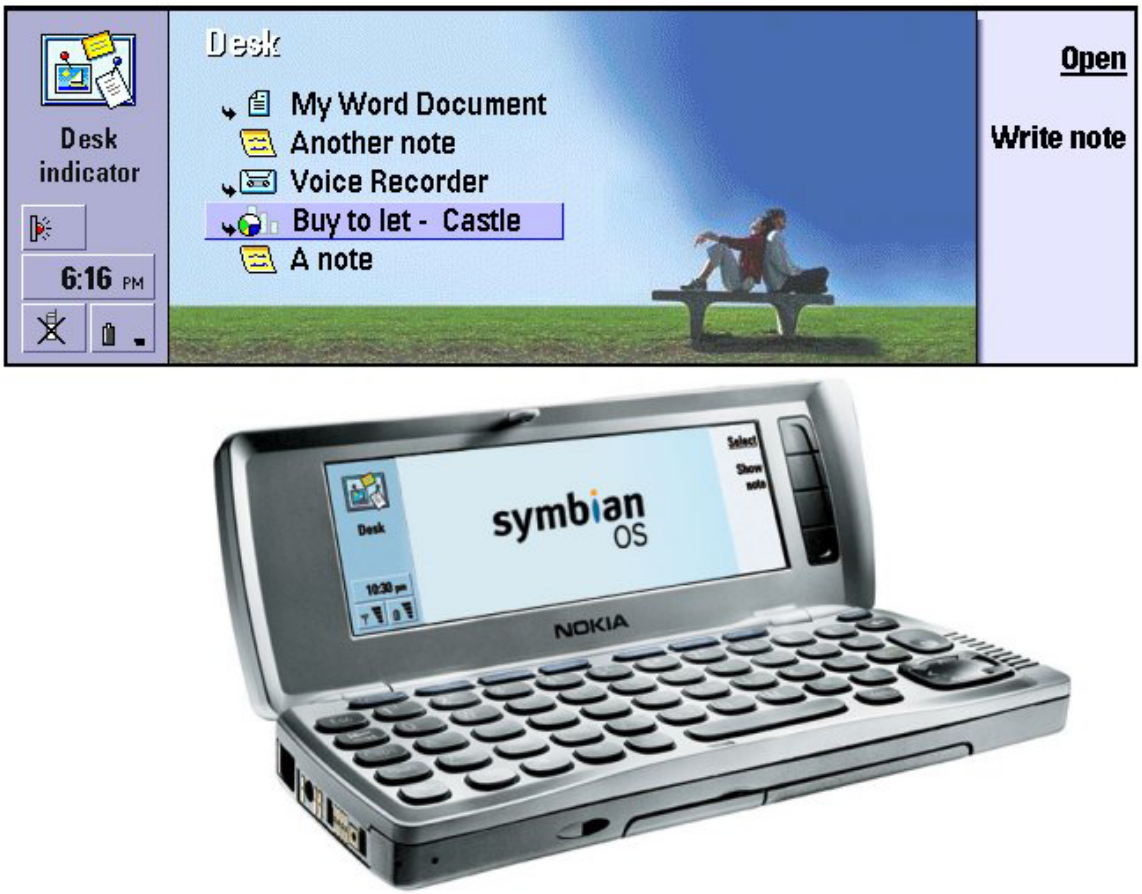


Figure 6: Crystal reference design for devices with full keyboard and large screen

2.4. Communications subsystem

2.4.1. Overview

Communications technologies can be divided into two main parts: transport technologies and content technologies / 8 / . Transport technologies are used for transferring data between communication peers. They involve communication media (physical component that is used to transfer data signals, for example, a Bluetooth chip, an infrared transmitter or a radio frequency module build on mobile phone) and communication protocols (rules for communications between devices). Content technologies are more clearly visible for

the end-user; these are e.g. messaging (email, SMS, MMS, etc) and WWW/WAP technologies.

One of the main characteristics of communication is that it works asynchronically. There is no guarantee how long it will take to transmit the data over the data link because environment and other aspects have effect on performance of used communication link. On this chapter, the main building blocks of Symbian OS communications subsystem are presented. Client-server architecture is handled first, because it will have a significant role in following communication systems.

2.4.2. Client-server framework

Client-server framework of Symbian OS is used for providing services for multiple applications that operates on different processes. Server offers services and handles resources that other applications use through client API. Many Symbian OS systems use client-server framework, for instance, Window Server, File Server and communications subsystem.

Process is the unit of memory protection and thread is the unit of execution. Each process contains one or several threads. Clients and servers exist in separate processes; the communication between clients and server is achieved by message passing or usage of inter-thread communication (ITC). Kernel server is used to route these messages correctly. The channel of communication between a client and a server is called as a session. One client can have multiple sessions open simultaneously with any servers. However, for efficiency reasons, often a client has only one session with a server, additional subsessions are then used for several communication channels. / 7 /

All services offered by a certain server are defined as an API that is presented as header (.H files) and library files (.LIB files) supplied by the vendor of the server. Client-server framework is safe to use because errors in clients don’t harm the functionality of servers. Accessing operating system's facilities using servers, instead of direct use of hardware

specific or deep-level services, is safe and convenient also from the application development point of view. For instance, clients don't need to know the exact hardware device or software protocols that are used to provide the required service.

2.4.3. Communications components

One of the key characteristics of Symbian OS communication components is the high level of dynamic extensibility. It means that a new hardware or new protocols can be added to operating system without restarting the device. This is provided by run-time loading of requested plug-in modules; these modules can be logical and physical device drivers (LDD, PDD), communication controls (CSY's), telephony modules (TSY's), protocols (PRT's) or message type modules (MTM's). / 8 / / 9 /

The major Symbian OS communications components and their main relationships are presented in Figure 7 / 6 / . The main servers: Serial comms server, Socket server, Telephony server and Messaging server are declared more accurately in following chapters. Figure shows also clearly the layers of Symbian OS communications subsystem, from applications to physical layer.

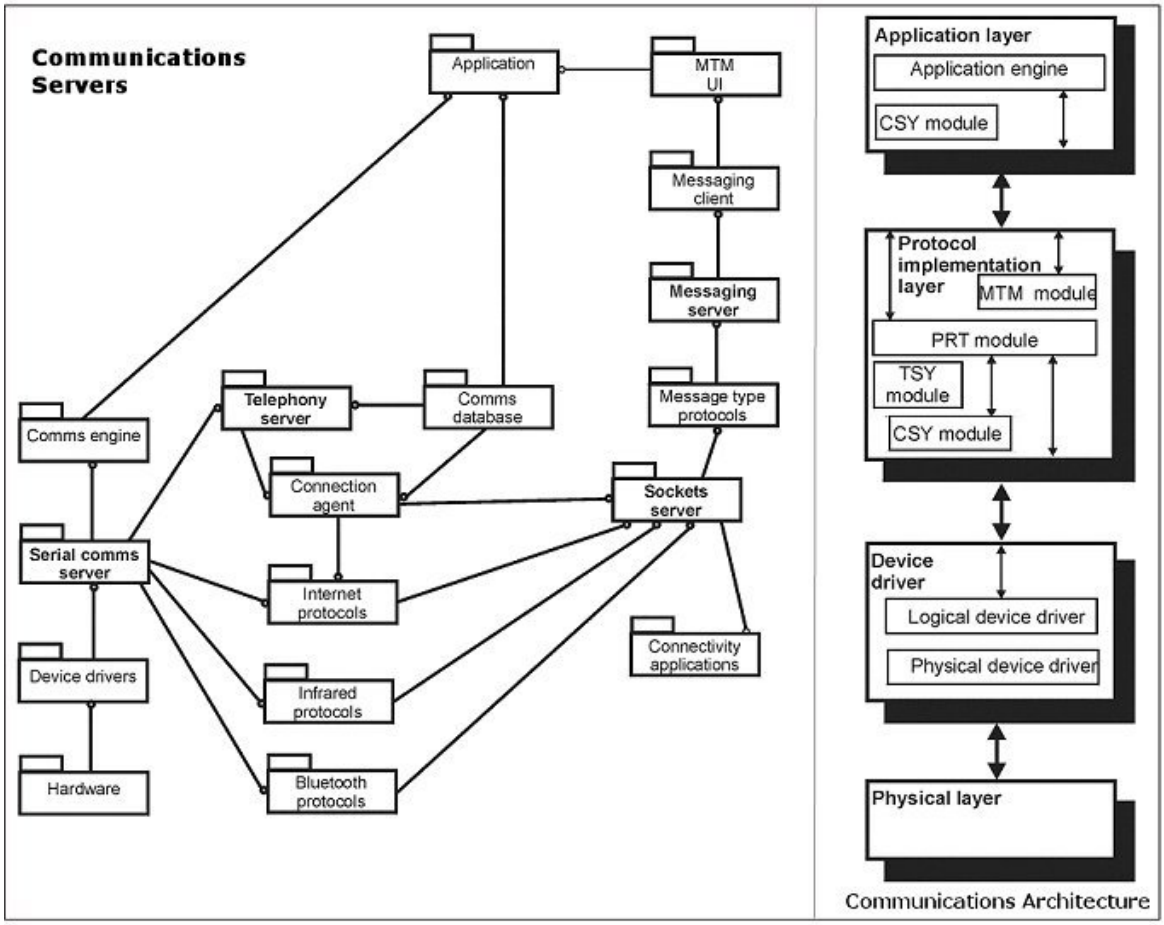


Figure 7: Symbian OS Communications subsystem

2.4.4. Serial communications server

Serial communications server (C32) offers a serial port API to its clients. This application programming interface provides services for any serial-like communication protocol to appear as a serial port in Symbian OS. Serial communications server is used to access the real serial communication over RS232 line and it provides so-called virtual interface, e.g. for infrared (IrDA) and Bluetooth communications. This "virtual interface" means an abstraction of a serial device that can be layered over any type of hardware.

To be able to use a serial port, the kernel has to be told to load a physical device driver (PDD), which talks to a specific hardware port, and a logical device driver (LDD) that

implements low-level port policy. These policies are, for example, flow control, buffer management and interrupt service routines (ISR) / 8 / / 9 / .

The serial service provider API consists of two abstract classes: `CSerial` (a serial service) and `CPort`, which presents an open and potentially shareable serial port / 9 / . Real serial services consist of dynamically loaded Communication server modules (CSY files). Normally, clients of Serial communications server (C32) don't need any information of individual communication server modules because it is asked to load active modules from Communications database of operating system.

2.4.5. Communications database

Symbian OS Communications database is implemented using DBMS (Database Management System) server and it is a central database for all communication related information. This database makes possible to implement device independent and easy-to-use communications components.

For example, an application that needs access to a serial service can consult the Comms database to find out the default serial communication module to load. This module can be, for instance, a standard RS232 serial line or a serial communication implementation of infrared protocol. The key advantage of this communication architecture is that there is no need for changing the original source code but all needed information can be loaded from Comms database.

2.4.6. Socket server

Symbian OS Socket server (ESOCK) provides a BSD-like (Berkeley Software Distribution) socket API to its clients. Protocol families are provided in specific protocol modules (PRT files), currently e.g. TCP/IP, WAP, SMS, infrared and Bluetooth protocol families are supported. / 8 / / 9 /

For instance, in Symbian OS, TCP/IP protocol family implements following protocols:

- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)
- Internet Protocol (IP)
- Internet Control Message Protocol (ICMP)
- Domain Name System (DNS)

All of these are defined in specific `tcpip.prt` file that can be loaded dynamically to be used as a part of Symbian OS application.

The main classes of Socket server are `RSocketServ`, which is a session to the server and `RSocket` that is the communication socket itself. `RHostResolver` is an interface for making host name resolution queries (e.g. inquiring Bluetooth device addresses) and `RNetDatabase` is an interface for network database access. / 10 /

2.4.7. Telephony server

ETEL is the Symbian OS Telephony server and it abstracts different kind of telecommunication devices into a common application programming interface (API) that is used by dynamically loaded telephony modules (TSY files). These devices could be, for instance, modems, phones or various mobile phones. Features of each of these devices are described in their own TSY files. / 8 / / 9 /

When Telephony server has loaded all available TSY's, a single client doesn't need any information about individual telephony modules. When a client application needs some telephony device, it requests the ETEL server to load the default TSY, which is specified in Comms database. ETEL is just responsible for telephony management, the clients control the actual transfer of information, and ETEL doesn’t participate in the data transferring through the connection it has established.

2.4.8. Messaging

The Messaging architecture provides a framework for creating advanced message client applications with plug-ins that support variety of messaging protocols. These modules are called Message Type Modules (MTM) and they wrap all needed functionality of lower level communication protocols, such as TCP/IP.

Dynamic extensibility of the Messaging architecture is provided by the run-time loading of Message Type Modules. As an example, a completely new mail protocol could be added to an existing Email application, e.g. IMAP4 mail protocol functionality could be added for using with earlier SMTP and POP3 features of the same application. / 8 / / 9 /

Message Type Modules aren't system wide components, so the information about them isn't stored in the Comms database. The Messaging system maintains four registers, which include lists of available MTM's and functions for loading and unloading the actual dynamic link libraries (DLL) that include the wanted Message Type Modules.

Symbian OS application framework and subsystems of operating system offer patterns that enable easier reuse. For example, as the socket interface is similar to different communication medias (TCP/IP, infrared etc) the reusability aspects can be considered during design and development of communication application. Next two chapters introduce software reusability, its technical and nontechnical aspects.

3. Software reuse

3.1. Introduction

Term reuse is defined in English dictionary in the following way: “To use again, especially after reclaiming and reprocessing”. Reuse in software development is also as vast area as described in earlier definition, although often it has been thought only reusing of old source code in new software projects. For example, Ivar Jacobson’s definition for software reuse is: Software artefacts are *designed* for use outside of their original context to create new systems / 11 / .

Therefore, software reuse effects on every part of software process and even whole software development organization as will be presented in following chapters. The overview for the main issues of software reuse is given in this and next chapter. In chapter three, the technical aspects of reuse are considered, from definition and architecture of reuse to production of reusable software artefacts. Next chapter handles business and process issues of reuse.

3.2. Scope of reuse

Reuse is often thought as reusing of the source code from previous projects using so-called copy-paste coding, i.e. code skeleton from older project is copied and modified for the use in new project. This method is also known as ad-hoc reuse. Software reuse is although much larger concept. In following table there are different point-of-views to reuse / 12 / :

Aspect of reuse	Definition	Example
Substance	What can be reused?	Ideas, concepts, artefacts, components, procedures, skills
Scope	The form and scope of reuse.	Vertical, domain-specific, horizontal, general-purpose, internal, external, small-scale, large-scale
Mode	Defines how reuse is executed	Planned, systematic, institutionalised, ad-hoc,

		opportunistic, individual
Technique	The approach that is used to implement reuse.	Compositional, generative
Intention	Defines how reusable elements will be used.	Black-box, white-box, glass-box, as-is, by adaptation, modified
Product	Defines what is reused.	Specification, architecture, design, source code, documentation

Table 2: Aspects of software reuse

3.3. Benefits of reuse

Today more and more demands are set to software development. New software products and upgraded versions of older ones should be on market more rapidly as ever before. Great expectations have been placed to software reuse for helping on these demanding tasks. Here are some of the apparent benefits of reuse in software development:

Quality

Reusable software components improve the overall quality of the software product. Developed components are tested and integrated to new applications several times and much more errors will be fixed comparing if the component would have been used just once. The reliability and performance of several times reused component is likely to get better. However, improved quality is achieved really only with reusable binary components. Copy-paste reusing can often have a certain effect that also bugs are copied to several software modules but the knowledge about their existence isn't spread to other developers. In this situation, the real quality improvement is often questionable.

Effort reduction

Software reuse reduces in most cases the specification, development and testing times of the software project. When using reusable components, there is no need to allocate resources to develop same functionality from scratch again and again. Same developer resources can be then allocated for other tasks that would need more developing efforts.

Reuse also reduces risks and costs of software maintenance and training. When several applications or components (=clients) use particular component, the risks and often also the costs of development of that component can be shared with all clients.

Several organizations in software industry have achieved significant gains in their reuse programs and long-term software projects. For instance, some organizations have gained big reduction in following metrics: time-to-market (reduction 2-5 times), defect density (reduction 5-10 times), cost of maintenance (reduction 5-10 times) and overall cost of software development (about 15 % but in some cases almost 75 % in long-term projects) / 11 / .

However, despite of these benefits several obstacles have prevented reuse to be as popular asset in software industry as it could be. Most of these obstacles are nontechnical and they are presented in more detail in chapter 4.

3.4. Layered architecture

Software architecture defines system structure, interfaces and interaction patterns of software under development / 11 / . Choosing the right architecture is often the most important decision in software development. It will have a significant role during whole software development cycle, from technical specification to development and even maintenance of software product.

The role of good architecture emphasizes when software should be designed for reused several times. Layered architecture (Figure 8) is a software architecture that organizes software in layers, where each layer is built on the top of another more general layer. Layer is defined as a set of subsystems with the same degree of generality. / 11 /

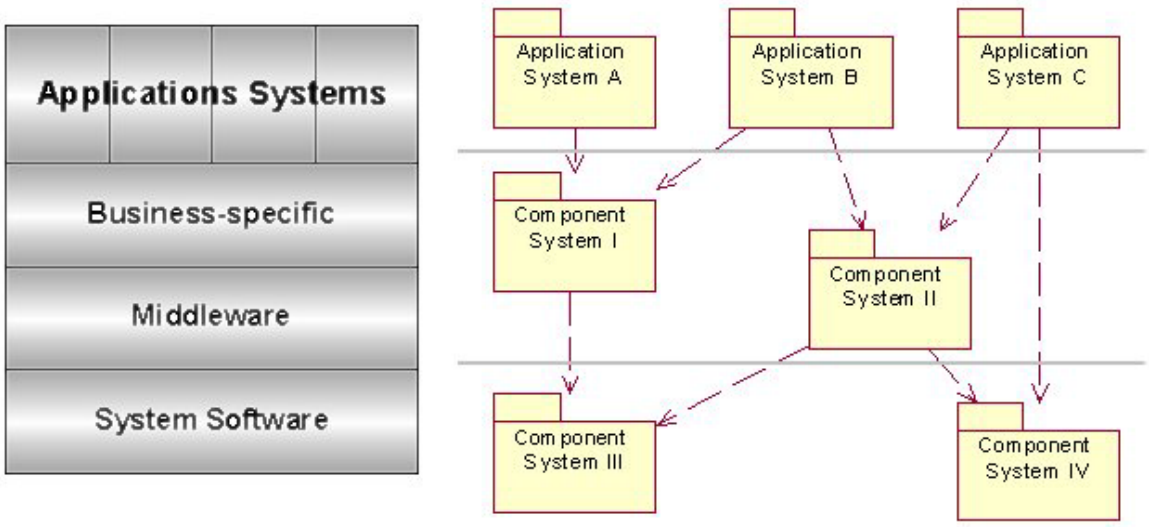


Figure 8: Layered architecture and an example of dependencies between layers

The upper layer, “Applications Systems”, contains application systems for particular end-user area. This end-user area could be, for example, office applications for personal computers or applications that operate with SIM card on mobile phone. For each of these end-user areas have been defined a set of use-cases that define common requirements to applications in particular application system (Application Family Engineering, AFE). For instance, a use case for office applications could be “User sends a document using e-mail”, this feature would then be implemented to several office applications (e.g. word processor, spreadsheet and presentation applications) using component systems from lower layers of architecture. Similar use case example for SIM applications system could be: “Read phone number from SIM card”.

Lower levels of architecture define component systems (Component System Engineering, CSE). Component system contains well-packaged set of components. Reuse engineers have developed them to be reused by application engineers or other reuse engineers / 11 / . Business-specific layer contains component systems that are specific to some business area. Application engineers use these components in their applications (Application System Engineering, ASE) and in optimal situation; component systems will enable rapid

end-user application development to particular application family. For example, the application framework of Symbian OS is a business-specific component system.

Middleware layer offers component systems that provide platform-independent services and utilies, e.g. interfaces to database management systems or object request brokers (e.g. CORBA) etc. The lowest layer, System software layer, contains component systems for networking, interfaces to special hardware, operating systems etc.

3.5. Software components

3.5.1. Definition

Reusable software components are self-contained, clearly identifiable artefacts that describe and/or perform specific functions and have clear interfaces, appropriate documentation and a defined reuse status / 12 / . Self-containedness and identifiability in previous definition mean that reusable component must be a clear and easily accessible entity that doesn't need great efforts to be reused. Reuse status means that component is maintained continuously; the re-user of the component knows who owns it and who will correct possible defects etc. This is very important, especially in large software organizations, where efficient communications between different departments of organization may be quite difficult.

Developing reusable components can be divided to two main parts: *developing with reuse* and *developing for reuse* / 12 / . Developing with reuse means that existing components are adapted and integrated to new context. Several integration cycles for these components are likely to make them more reliable and efficient. Developing for reuse means that components are designed for reusing in other context than they were initially developed / 12 / . Design for reuse is a very demanding task that needs software specialists that have adequate experience of software development. Otherwise, the scope of the developed reusable component may be too narrow and it can be utilized only once; then the real benefits of reuse aren't achieved at all.

3.5.2. Component models

Reusable components form the basis for the development of application systems. Several component libraries have been built to fasten software development in some particular domain. Sun's Enterprise JavaBeans (EJB) is a Java-based server-side component library that can be used for developing large distributed applications / 15 / .

Microsoft's Component Object Model (COM) is a component software architecture that allows systems to be built from components supplied by different software vendors. / 16 / Object Management Group's (OMG) open Common Object Request Broker Architecture (CORBA) is a vendor-independent architecture and infrastructure that computer applications can use to work together over different networks. A CORBA-based program from any vendor, on almost any computer, operating system, programming language and network can interoperate with a CORBA-based program from the same or another vendor. / 17 /

3.5.3. Symbian ECOM architecture

Symbian published a new architecture model for generic interfaces (Modular Architecture for Generic Interface Components, MAGIC) in the newest version of operating system. This model is called ECOM and it provides a generic framework for registering and discovering appropriate interface implementations at run-time. / 18 /

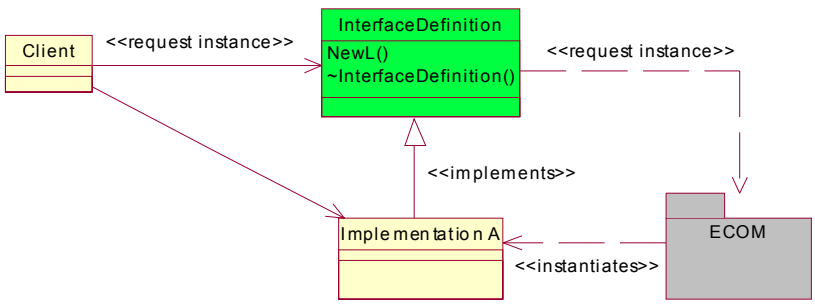


Figure 9: Symbian ECOM architecture

Figure 9 is an example of ECOM architecture. A client wants to use some implementation of particular interface. Design principles (introduced in chapter 3.7) state that software module should be depended only on abstractions (=interface), not concrete implementations. Traditionally, in order to avoid client's dependency on the actual interface implementation, some design pattern (e.g. Factory pattern, example on chapter 6.4.3) is needed for instantiating the concrete implementation for the client. Implementing this principle correctly and avoiding unnecessary dependencies may be difficult even for the experienced developer.

ECOM architecture simplifies previous procedure significantly. A client uses concrete implementation through abstract interface, but ECOM architecture is responsible for instantiating appropriate implementation. Client only specifies the wanted implementation with `NewL` method of interface.

3.6. Commonality and variability analysis

Efficient design of component systems needs analysing of commonality and variability inside particular system. Commonality and variability mean features that are common and variable between different components. This is illustrated in Figure 10, there are three different applications (or components) and colours are used to define common features. Darker grey means that these features are common for all three applications; lighter grey means that two but not three applications have this common feature. White area describes variable features that are individual for each application in the application family.

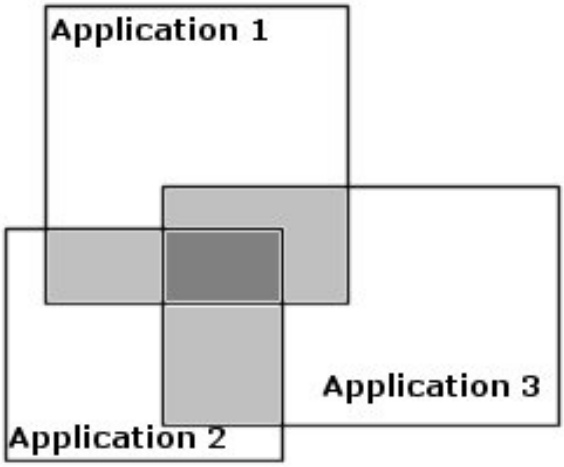


Figure 10: Common and variable features of three applications in same application family

James Coplien has researched scope, commonality and variability (SCV) analysis. The mains steps of analysis are / 19 / :

1. Define domain and scope for the component.
2. Identify commonalities and variabilities of it.
3. Set appropriate limits for variabilities. This is necessary because without limits the component would be too difficult or even impossible to design and implement. In addition, the performance of component "that-will-do-everything" might not be very good.
4. Exploit commonalities and adapt variable functionality.

Step 4 in SCV analysis is the starting point for designing the actual software component. Design principles and patterns are briefly introduced in following chapters. However, the big picture can be seen in Figure 11 / 14 / . Common and variable features affect on perspective of software development process: architecture, design and implementation.

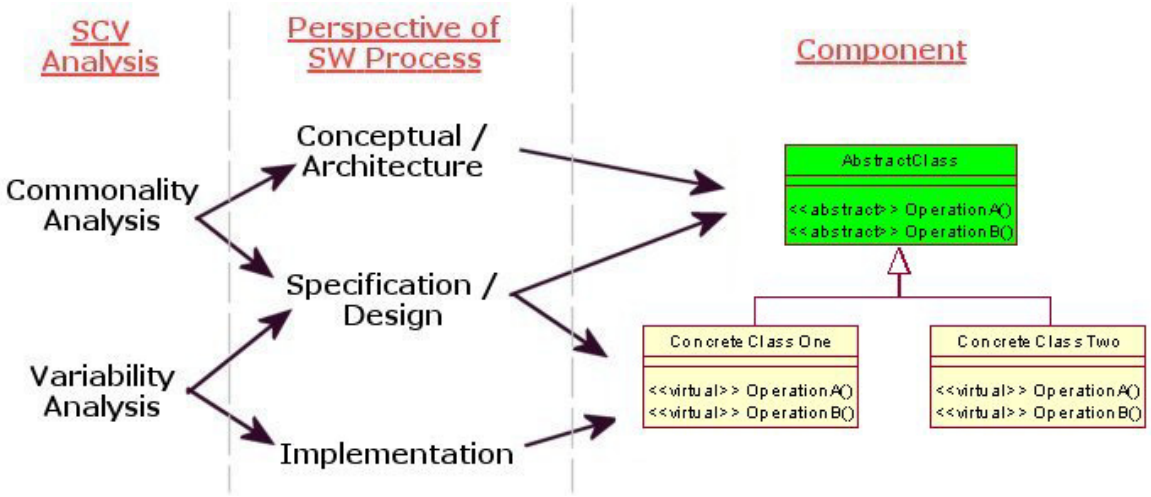


Figure 11: Commonality and variability analysis

3.7. Design principles

Software must be designed to be reusable. Object-oriented technology itself doesn't add value for developing reusable software. Design principles and patterns based on SCV analysis are the main driver for developing software that is later possible to reuse in other application systems.

Design principles describe fundamental principles of object-oriented design. Several principles have been introduced in literature, but here is a brief introduction to most important ones:

1. Open-Closed Principle (OCP): *A module should be open for extension but closed for modification*
2. Dependency Inversion Principle (DIP): *Depend upon Abstractions. Do not depend upon concretions* / 26 /

These two principles are different views to same issue. At first, Open-Closed principle may sound conflicting, but it means that when some module is closed for modification there is no need for testing already tested features again. Open for extension means that new features can be added to module without interfering the old implementation. How is this

possible? The second principle is the key for this; a Dependency Inversion Principle states that interface classes should be used as much as possible. In other words, use abstractions, not concrete instances of classes. Inheritance in object-orient programming is a concrete way to implement these principles. Abstract base classes define the interface that dependent classes use and inherited concrete classes implement it. DIP is also known as "*design to interfaces*" principle.

3. The Acyclic Dependencies Principle (ADP): *The dependencies between packages must not form cycles* / 26 /

Figure 12 shows the real meaning of Acyclic Dependencies Principle. Component System IV makes a cyclic dependency with Component System I. This kind of dependencies should be always avoided, because the maintenance and the further-development of the system become very difficult. Without cyclic dependency the Component System II could be developed and tested independently but with current dependencies every system is dependent on each other.

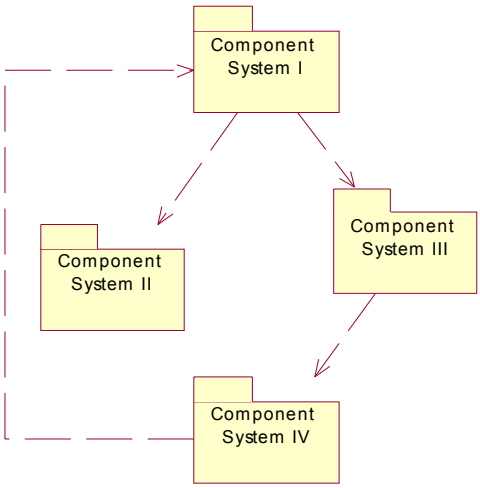


Figure 12: Example of cyclic dependency

4. *Favour composition over inheritance / Avoid strong dependencies* / 25 /

Composition means that objects are composed and the relationships between them are done dynamically at run-time as inheritance relationship is defined already when the code is

compiled. In practise, composition is implemented following way: class A contains pointer to interface class MB, when class A is instantiated it gets pointer to class CB as a parameter. Class CB is a concrete implementation of interface defined with class MB.

Implementing class relationships with inheritance causes sometimes too heavy class hierarchies and maintaining them may be difficult. Favouring object composition over class inheritance keeps each class encapsulated and focused on one task. Classes and class hierarchies will remain small and more maintainable. / 25 /

This principle could be stated also as “favour low coupling and high cohesion”. Coupling is a measure of interconnections among software modules and cohesion is a measure of the relative functional strength of module / 29 / . Low coupling means that there should be dependencies between modules as few as possible and high cohesion means that one module should have functionality as few as possible. However, design of software system needs always trade-offs, so the system that is “low coupled and high cohesive” is very difficult or sometimes even impossible to design.

3.8. Design patterns

Design principles are quite simple rules, but in practice, it needs quite significant design efforts to implement these rules correctly. Design patterns are customized to solve a general design problem in a particular context / 25 / . They give a time-tested higher-level perspective to problem domain, provide reusable models for designing object-oriented systems and establish a common terminology for the development team.

Design patterns are often divided to three groups: creational, structural and behavioural patterns / 25 / . Creational patterns are used for designing the creating of objects in a system. Structural patterns give methods for dealing with relationships of objects and behavioural patterns provide patterns for interactions and responsibilities of objects.

Several design patterns are presented, for example, in "Design Patterns: Elements of Reusable Object-Oriented Software" book / 25 / but here is one example of structural patterns (Adapter pattern in Table 3). Couple of more design patterns are described in chapter 6.4.3, where the design patterns of developed Bluetooth component are presented.

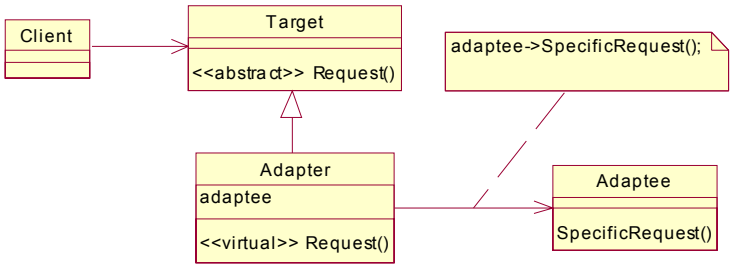
Pattern	Description
Adapter	Convert the interface of a class into another interface that clients expect. Adapter lets classes to work together that couldn't otherwise because of incompatible interfaces.  / 25 /

Table 3: Example of design pattern: Adapter

3.9. Component production

Idea for software component rises often on software project where some module has been developed and there is obvious need for the same functionality in future projects. In order to transform developed module into a reusable component, it needs generalization based on SCV analysis (see chapter 3.6).

Components change significantly the software development paradigm, the code isn't developed from scratch but more and more of software implementation is integration of reusable components. Operating system, 3rd party and customer-specific components are integrated to software product. Benefits of component integration are evident; development cycles shorten, defect rates are likely to decrease and development resources can be allocated to other demanding tasks. However, using components have also some

disadvantages, for instance, project management becomes more difficult, especially when components from third party developers are used. Human factors may effect on development with components that are initially developed somewhere else (so called “not invented here” syndrome). There is more information available about these obstacles on chapter 4.2.

4. Reuse business

4.1. Introduction

Software reuse as an idea sounds excellent and one might wonder why it hasn't spread all over the software development. The reason for this is sometimes technical; software domain and very long-term projects may have a certain effect that the software development organization doesn't see the evident benefits of reuse. Especially if software project continues for several years and the next project that organization starts is distinctly different, the common functionality that could be potentially reused may be hard to find. However, when software reuse hasn't been taken as a part of daily work of development organization the reason is often nontechnical.

4.2. Obstacles of reuse

Software reuse in several software development organizations has been ad-hoc reusing, which is often the earliest step of reuse. Some source code is reused with copy-paste method and some designs may be also utilized again but no systematic approach is used. Often in this kind of situation, possible resistance for reuse occurs in every party of software development projects.

Software developers may suffer from "not-developed-here" syndrome / 12 / , which means that they don't completely trust software components that have been developed somewhere else. Developers think that usage of component from some other developer limits their creativity and they think that they could be able to develop "much better component faster" than the other developers could.

Software project managers are often in difficult situation, they are responsible of projects to customers, upper management of the company and members of project team. Their projects should be ready on agreed schedule, budget and resources. Developing reusable components in this kind of situation is often less important and only components that fit

only that particular project are developed. This behaviour could be called as “not-in-my-project” syndrome / 12 / . Customer isn’t likely to support developing of reusable components in its project, especially if the need for same kind of functionality isn’t apparent in the near-future projects of the same customer. Therefore, developing this kind of components further to reusable ones needs new development projects. This needs support from upper management and sadly often further-development projects may be impossible due to lack of appropriate resources and budget.

Upper management of software company should be always concerned about the long-term success of the company and its customers, partners and employees. However, especially young software organisations often have problems starting efficient software development business. Starting systematic software reuse program at this point is often unrealistic because the biggest concern is to get couple of projects to start and long-term view for software reuse isn’t the biggest problem during daily work.

Lack of upper management support may be a problem also later when software development organization has developed processes for daily work and they have a solid project track for continuous working. At this point, the lack of upper management support may appear e.g. in inefficient budgeting or resource allocation for software development / 12 / . The importance of decent software development practices is concerned but the reality may although be different.

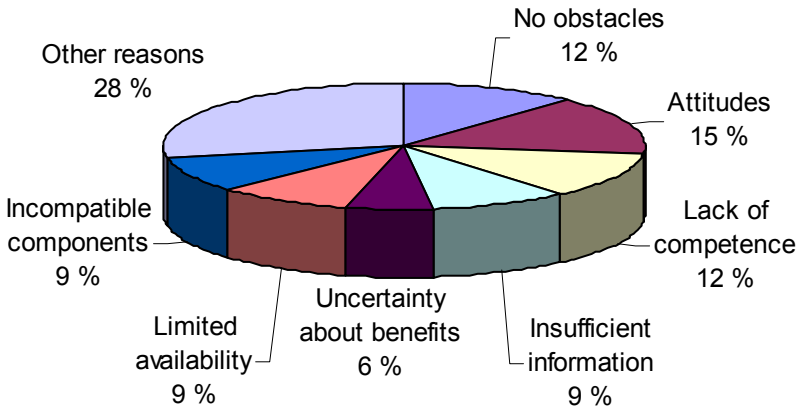


Figure 13: Obstacles to the exploitation of software components

Figure 13 shows other obstacles that prevent the usage of reusable components / 13 / . Lack of competence, insufficient information and uncertainty about benefits are all reasons that can be avoided quite easily. Training and workshops about reusability and components for whole organization will have significant positive effects on starting component based software development program. Limited availability and incompatible components are problems that can be solved later as reusable software artefacts and reuse processes mature in organization.

All previously mentioned problems need to be solved for that reuse program could work efficiently in software organization. Some motivation for establishing reuse program is presented in following chapter, which handles the economics and processes of software reuse.

4.3. Economics of reuse

Despite of obvious technical and project management benefits of starting reuse program in the organization, the only thing that matters in the long-term success of the company is money. Being part of reuse the business means that reusable software artefacts are used to shorten developing cycles and time-to-market of software products and this way to save the money of the company. As all product development activities, also software reuse program is an investment for the future. Investments bind money for some period of time and only in successful case that particular investment will start to profit and earn interest for invested money.

Efficient management of organizations and business needs some concrete input, measures, that could be used as part of justified decision-making. Several technical metrics for reusable components can be defined, couple of examples can be found on chapter 6.5.4. Economical model for reuse can be arranged to three phases / 11 / :

1. Measurement
   Measures are based on previous software projects; what kind of reuse level has been achieved before, resources and costs of previous projects.

2. Cost / Benefit estimation
   Measures are related with each other and with the basic resources of all software projects: effort, cost and time.

3. Reuse investment analysis
   Measures and estimates are analysed for determining how the reuse business of software organization is really doing and what corrections should be done.

Developing reusable software needs much more effort than ordinary “let’s code this module just for this application” method. The extra costs needed for this must be paid somehow, the basic methods are:

a) Decreasing costs of future projects by the usage of reusable components. It is affordable to use reusable component so always developing a new one in future projects isn't an option.
b) Shortening significantly time-to-market of software product and possible profit from sales would pay back the investments.
c) Reusable components can be sold to other software development companies.

Successful investments for software reuse programs have been noticed to be very domain, business goal and long-term customer relationship specific. As a rule of thumb, in order to get developing of reusable component to be profitable, at least three integration rounds or application engineering cycles must be done before payback of investment could start / 11 / .

4.4. Software reuse processes

4.4.1. AFE, CSE, ASE processes

Ivar Jacobson defines software reuse processes based on the layered architecture defined in chapter 3.4. Designing software for reuse needs clear processes and responsibilities to work efficiently. Essential steps for Application Family Engineering, Component System Engineering and Application System Engineering are same, only the focus is slightly different on each area. The processes presented here are quite heavy because they are intended for developing several applications and component systems within large application families. However, processes are adaptable also for lighter software development. The main process steps are presented here:

Requirements	
AFE	Application Family Engineering develops and maintains the overall layered system architecture / 11 / . The first step is to select the scope for the application family according to markets, customer and end-user demands and business goals of the company. Requirements from previous parties are gathered and iterated and the most important ones are described as use cases for the application family. The main

	issue is to find features that are common for all members of application family.
CSE	Component System Engineering focuses on building and packaging robust and extensible components / 11 / . Requirements for component system are collected with focus on variability. This is far more difficult than in AFE process. Features of components should be used in several applications and they may be used for several years. The main issue is to understand the variability of components and what kind of trade-offs must be done for enabling it.
ASE	Application System Engineering is so-called ordinary software development, so gathering requirements is often the first part of software project. Important part is to try to map application requirements for component use-cases and to find out what kind of components may be needed more and how current ones should be adapted.
Robustness analysis	
AFE, CSE, ASE	Use cases produced from requirements gathering are used for making high-level analysis model. Analysis model consists of analysis objects, subsystems and their relationships and they define the architecture of the system / 11 / .
Design	
AFE	Analysis model is used for designing prototype for the layered model that would include possible applications and component systems. Subsystems should be quite small because it allows easier changes and makes layered system more manageable.
CSE, ASE	Requirements and analysis model / architecture are the basis for the design of components and applications.
Implementation	
AFE	Implementation of application families must be done systematically. Architecture of each subsystem must be iterated according to commonalities and variabilities of the application family.
CSE, ASE	Component systems and applications are implemented.
Testing	
AFE, CSE, ASE	Component and application systems are tested and the overall performance and architecture scalability are considered from the application family point of view.
Packaging	
CSE, ASE	Components are documented and packaged for as easy as possible reusing. Application system is packaged and documented for use by the manufacturers and end-users.

Table 4: Process model for software reuse

4.4.2. Reuse organization

Establishing reuse program means also setting appropriate reuse organization. Figure 14 / 11 / shows some roles in reuse organization. Customers and end-users set the initial requirements for software products. Division has been done for Application System Engineering and Component System Engineering and there are several different roles for both of them. Often same person may have several roles, but in large software development organization, each role has its own responsible person.

Use case engineers develop use case models based on input from end-users, customers and reusers. Application and component engineers design and develop applications and components and testers test them. The role of architect is essential in software reuse organization. Architects are responsible for high-level architecture of application families, single applications or component systems.

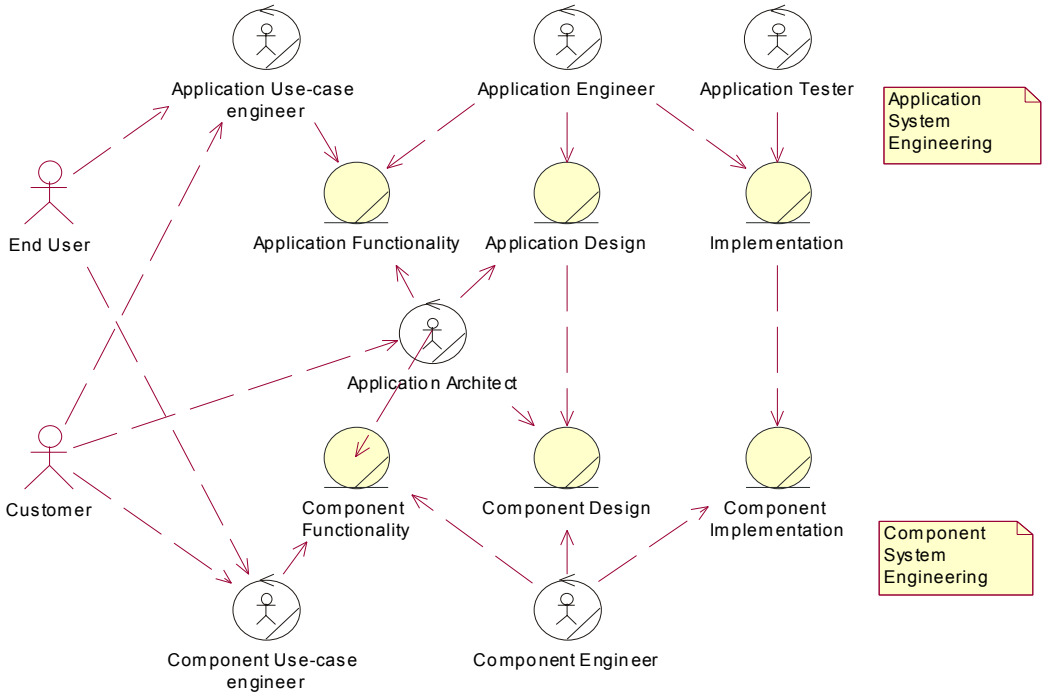


Figure 14: Organization for reuse business

4.4.3. Adaptation of reuse program

Starting reuse program in software organization isn't an easy task and nothing happens in one night. Reuse processes must be adapted into daily work with little steps. Big and significant changes in one step are likely to cause problems. For instance, daily software development may interfere too much and resistance may appear among developers.

Figure 15 presents one example about incremental steps of starting reuse program. At first, reuse is just informal code reuse, but as processes mature development transfers to black-box reuse and finally to domain-specific reuse-driven software development. However, the time scale depends heavily on the software domain, maturity of processes and conforming to them. In addition, the experience of workers has a significant effect to the length of time scale.

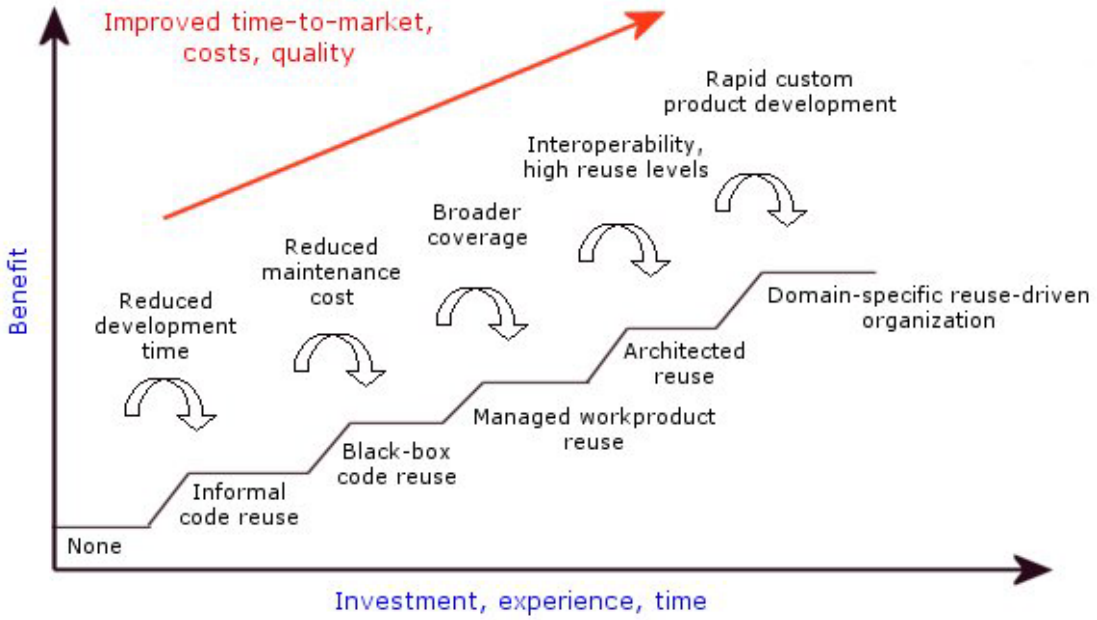


Figure 15: Incremental adaptation of reuse

4.4.4. Digia Software Process and reuse

Digia Software Process (DSP) is an incremental and iterative software development process / 1 / . It is used as a guideline for daily work in Digia's customer and own product development projects. The main building blocks of DSP are presented in Figure 16.

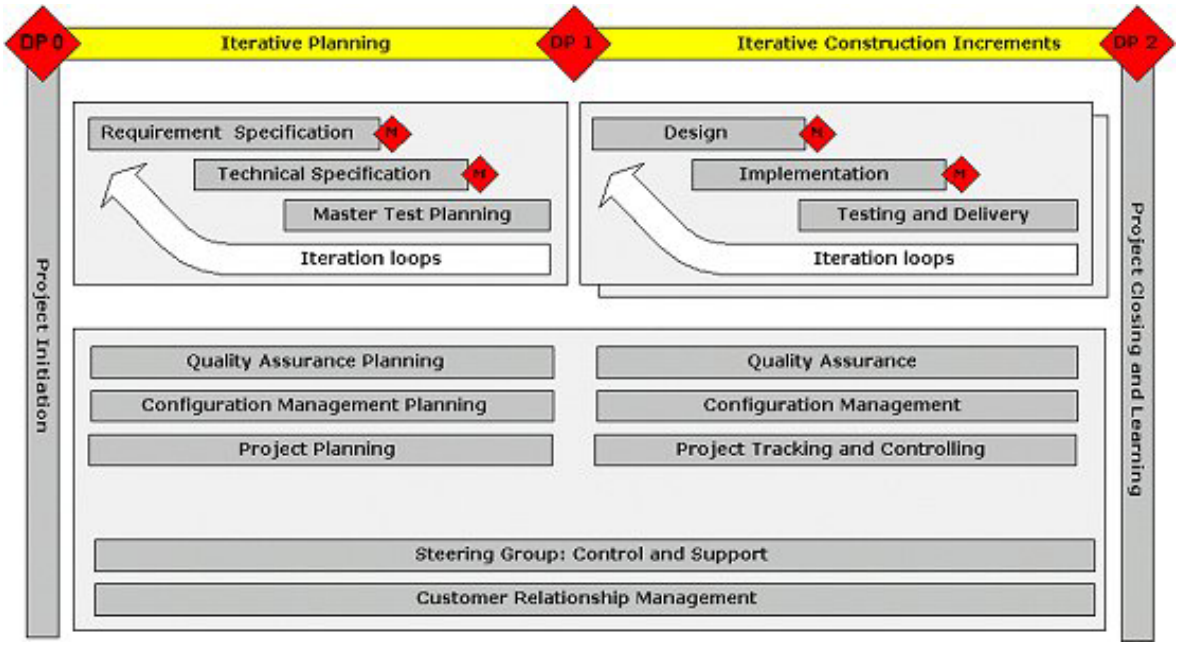


Figure 16: Digia Software Process

During the first half of year 2002, Digia started to build up its own reuse program for software development. The main steps of Digia's reuse processes are:

1. In the beginning of software project, needed components are identified. Some of these components have already been made and must be adapted to use in particular project. Some components are done from scratch during the project. Individual developers are also encouraged to component ideation.
2. Component Management Board (CMB) analyses, processes and prioritises component ideas. It maintains also long-term roadmap for the component development.
3. As CMB approves the development of some component, it is developed in product development project or as an individual component project.
4. When component is ready, it is reviewed, approved and stored into component library to wait for reuse.

This and previous chapter contained the overview of software reuse and software components. Practical work of this thesis was to design and implement a Symbian OS based reusable software component for Bluetooth networking. This issue is handled on chapter 6, but before that, the next chapter gives an overview to Bluetooth technology.

5. Bluetooth technology

5.1. Introduction

Bluetooth is a short-range radio link between different kinds of communication devices, such as mobile phones, computers, PDAs etc. Wireless Bluetooth technology has been developed by Bluetooth Special Interest Group (SIG) that has several thousands member companies (Ericsson, Nokia, 3Com, IBM, Intel, Microsoft etc).

The main idea of Bluetooth technology is to allow communication between devices to be as simple and cheap as possible. Especially the cheapness of Bluetooth chip will be the main driver to extend this radio technology everywhere.

5.1.1. Bluetooth - Peer-to-Peer communications technology

Bluetooth is a promising technology for local peer-to-peer communications. The nature of peer-to-peer defines that each party has an equal role in network, so the traditional slave-master / client-server division isn't so important. However, these names are used for clarifying device roles, for example, in connection establishment and managing timing and frequency hopping of devices. In addition, Bluetooth application developer may find it easier to manage the Bluetooth network when one device is defined as a master device of the logical network. Bluetooth specification defines that the master is the device that starts connection establishment / 23 / .

Local Bluetooth network is called as a piconet. A piconet contains always one master device and 1-7 slave devices. A single Bluetooth device can act in both roles, as a master or a slave, and this way construct several piconets, called scatternets. Scatternet topology is especially used if there is need for Bluetooth network with more than 8 (1+7) devices.

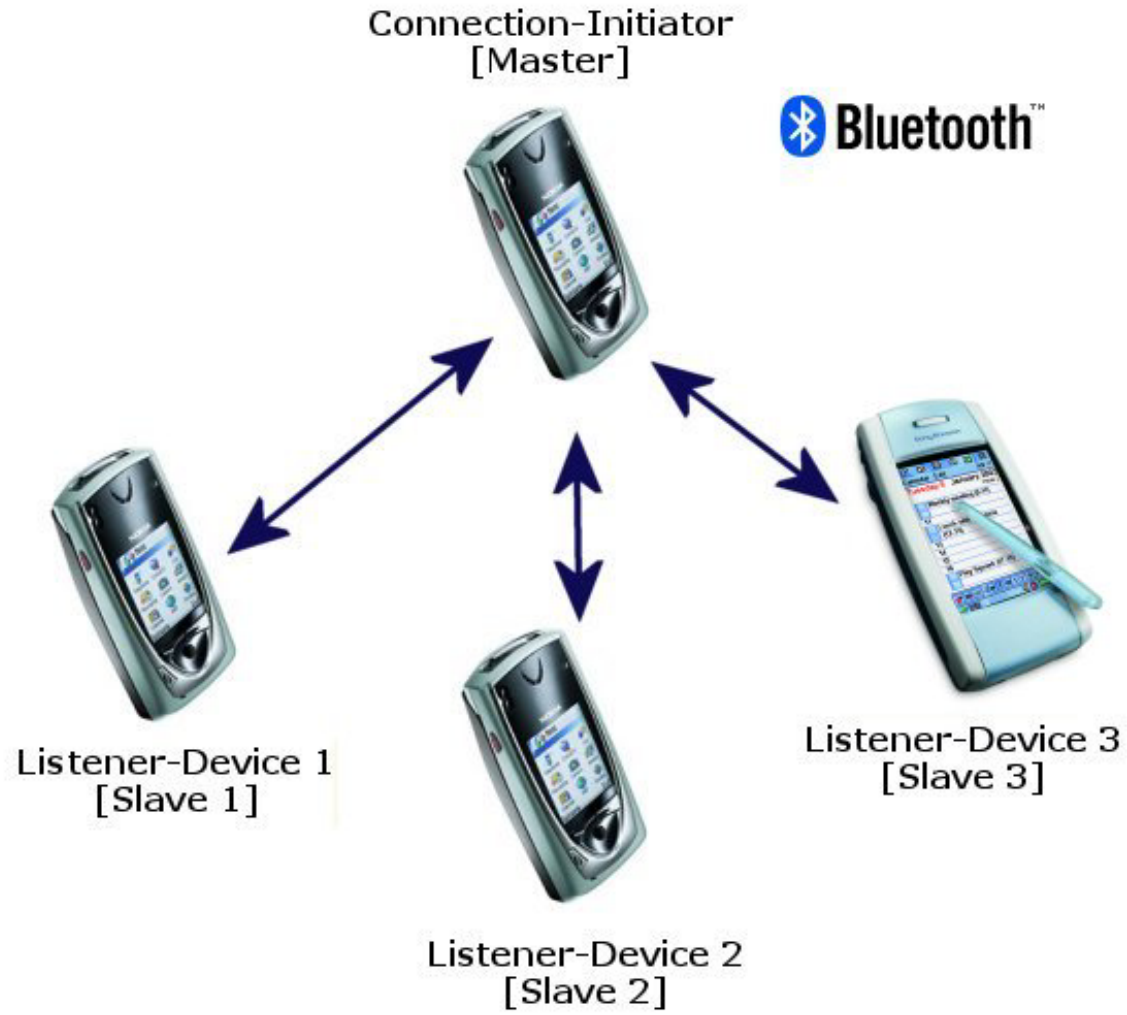


Figure 17: Bluetooth piconet

Bluetooth connections must be made always with known devices. Before connection can be established, a device must inquire other local devices to get their device addresses. User can define a Bluetooth Device name (e.g. “Jack’s BT Phone”) for his device, which is often more informative for end-users than just device address (long hexadecimal address e.g. 0x00e00379b874).

5.1.2. Service discovery

After successful device inquiry, a service discovery must be made with selected devices. Service discovery is essential because end-user cannot conclude anything from BT device address or even device name, the inquired device could be a mobile phone, a laptop computer or even a Bluetooth-enabled car tire / 30 / .

Each Bluetooth device has a service database, which includes all Bluetooth services that are available in that particular device. Service database may contain 0-N service records. Each record has a well-known ID number that is used for detecting appropriate service record in the remote device, these ID numbers are defined in BT specification / 23 / . One service record contains 1-N service specific attributes.

Service (e.g. a file transfer, mobile game, etc.) registration is needed with 'listening-side' devices before connection establishing. The connection-initiating device will discover these service records and tries to find appropriate service record using service ID. After that, it resolves needed attributes (e.g. communications channel, service version etc.) for successful connection establishment.

5.1.3. Connection establishment

The fastest and most logical way to establish a Bluetooth network with unknown devices is that one of the users selects his device to be a connection-initiator (master in BT network) and all other devices are listener-devices (clients). A connection-initiator will inquire all devices and discover service for user-selected ones. After successful service discovery, data connection to selected devices can be established and data can be sent and received.

If Bluetooth device address is already known (e.g. BT address of friend's smartphone is stored in your own device after previous session) the connection establishment procedure is clearly faster because the service discovery can be started immediately.

5.2. Bluetooth protocol stack

The Bluetooth protocol stack is presented in Figure 18. During the specification phase of new Bluetooth enabled device, several decisions must be made according to the requirements (cost, time-to-market, features, etc.) of new product. For instance, what kind of Bluetooth chip and stack will be used in product and how will be the stack software stored between BT chip and host device.

There are three basic models for implementing stack: hosted, embedded and fully embedded / 21 / . Symbian OS and e.g. Nokia 7650 smartphone uses hosted Bluetooth stack implementation. This means that the lower part of the stack is on Bluetooth chip set of the device, the upper part and all BT applications are on the host device. Communication between host and chipset are done using Host Controller Interface (HCI). This implementation model enables changing BT hardware without interfering upper layers of protocol stack or applications of the host device.

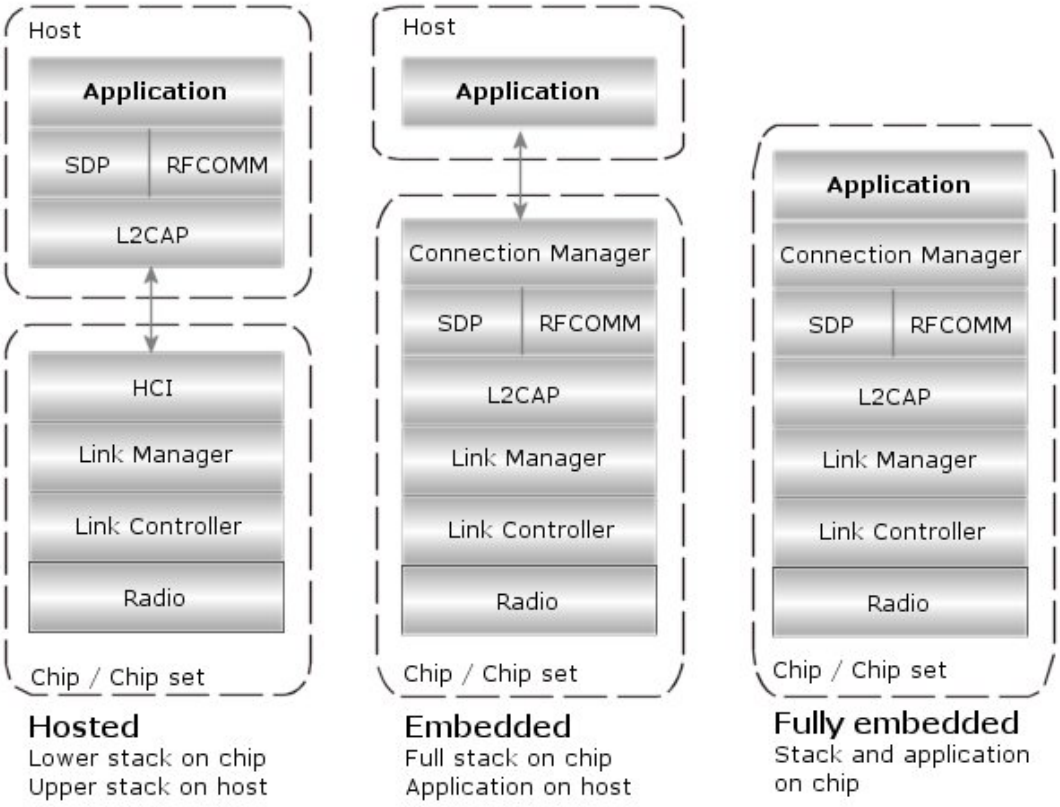


Figure 18: Models for Bluetooth stack implementation

5.2.1. Radio, Baseband and Link Control Layer

Bluetooth technology uses 2,4 GHz ISM band for its radio link. It utilizes fast frequency hopping scheme with error protection and correction for avoiding interference of other systems operating on same ISM frequency band. In Frequency-Hopping-Spread-Spectrum (FHSS) system, packets are transmitted in defined time slots on defined frequencies. Bluetooth uses Gaussian Frequency Shift Keying (GFSK) method for modulation of radio signal./ 20 /

Baseband layer uses inquiry and paging procedures to synchronize the transmission hopping frequency and clock of different Bluetooth devices / 20 / . Baseband layer provides two kinds of physical links, Synchronous Connection-Oriented (SCO) and Asynchronous Connectionless (ACL). ACL packets are used for data only, SCO packets may contain also audio or combination of both of them.

5.2.2. Link Manager Protocol

The Link Manager Protocol (LMP) is responsible for link set-up between Bluetooth devices. Layer includes security aspects like authentication and encryption by generating, exchanging and checking link and encryption keys. LMP layer controls also the power modes of the BT radio device and connection states of a BT unit in a piconet.

5.2.3. Host Controller Interface

Host Controller Interface (HCI) is used in hosted implementation model of protocol stack (lower layers in Bluetooth module and upper layers in host device). It provides standardised interface between host and BT module, and this enables interoperability of different hosts and Bluetooth modules of different manufacturers.

5.2.4. Logical Link Control and Adaptation Protocol

The Bluetooth Logical Link Control and Adaptation Protocol (L2CAP) adapts the upper layer protocols over the baseband. It provides connection-oriented and connectionless data services to the upper layer protocols with protocol multiplexing capability, segmentation, and reassembly operation. / 20 /

5.2.5. RFCOMM

RFCOMM is a serial line emulation protocol, which emulates RS-232 control and data signals over Bluetooth baseband. RFCOMM provides transport capabilities for upper level services (e.g. Object Exchange protocol, OBEX) that use serial line as a transport mechanism.

5.2.6. Service Discovery Protocol

Bluetooth is very service-centric technology as mentioned on chapter 5.1.2, not device-centric as personal computers in traditional local area networks. Service Discovery Protocol (SDP) is very important part of the Bluetooth framework. Bluetooth device has SDP database where information about all services is stored. When Service Discovery Protocol is used, device information, services and the characteristics of the services can be queried from service database and a connection between two or more BT devices can be established.

Figure 19 / 23 / shows service record for file transfer profile. The four main attributes (red rectangle around them) must be registered to local Bluetooth service database that application, which supports BT file transfer profile would work correctly. Each attribute can have several different value types (several kinds of signed and unsigned integers, Boolean values, strings etc) or it can be a list, which is recursively built from other attributes or lists.

Item	Definition:	Type/ Size:	Value:*	AttrID	Status	Default Value
Service Class ID List				See [15]	M	
Service Class #0		UUID	OBEX-File Transfer		M	
Protocol Descriptor list				See [15]	M	
Protocol ID #0		UUID	L2CAP		M	
Protocol ID #1		UUID	RFCOMM		M	
Param #0	CHANNEL	Uint8	Varies		M	
Protocol ID #2		UUID	OBEX		M	
Service name	Display-able Text name	String	Varies	See [15]	O	"OBEX File Trans-fer"
BluetoothProfileDe-scriptorList				See [15]	O	
Profile ID #0	Supported profile	UUID	OBEX File-Transfer			OBEX File Transfer [15]
Param #0	Profile ver-sion	uint16	0x100			0x100

Table 6.1: File Transfer Service Record

Figure 19: Example of service record in service database

5.3. Bluetooth profiles

Bluetooth profiles provide a clear description of how a full specification of a standard system should be used to implement a given end-user function / 20 / . In practice, this means that profiles define common interface for all Bluetooth devices that want to conform to the particular profile. Bluetooth SIG has published profiles specification / 23 / in their website (http://www.bluetooth.com/). It describes several profiles and their interfaces (e.g. Headset, FAX, LAN access point, File transfer profiles etc).

When a common interface is defined, Bluetooth devices from different manufacturers can interoperate with each other. For example, a user wants to read his emails with his laptop when he is waiting in railway station. A laptop from manufacturer A needs to work with a mobile phone from manufacturer B. Both devices implement a Dial-up networking profile / 23 / that is used for establishing dial-up connections to remote computers. Laptop uses Dial-up information of the mobile phone and establishes Bluetooth connection to it. Mobile phone knows what to do when a BT Dial-up connection is established, in this case, it makes a GPRS connection to the email server of the service provider and fetches all new emails.

In practice, the attributes that are defined for a profile are stored to the service database of Bluetooth device. When other BT device is discovering services that would fulfil the needs of this profile, it knows what attributes should be found and knows how to use them.

5.4. Bluetooth in Symbian OS

Symbian OS has supported Bluetooth technology since the version 6.0 of operating system. In the Symbian OS Bluetooth Architecture, the core stack functionality is implemented by two components (Figure 20, / 6 /), Host Controller Interface (HCI.DLL) and the Bluetooth Protocol module (BT.PRT). The Host Controller Interface module encapsulates the set of BT HCI commands and events. Bluetooth Protocol Module (BT.PRT) encapsulates L2CAP and RFCOMM layers. As a Symbian OS protocol module, it provides a Socket API to these protocols. / 8 / / 22 /

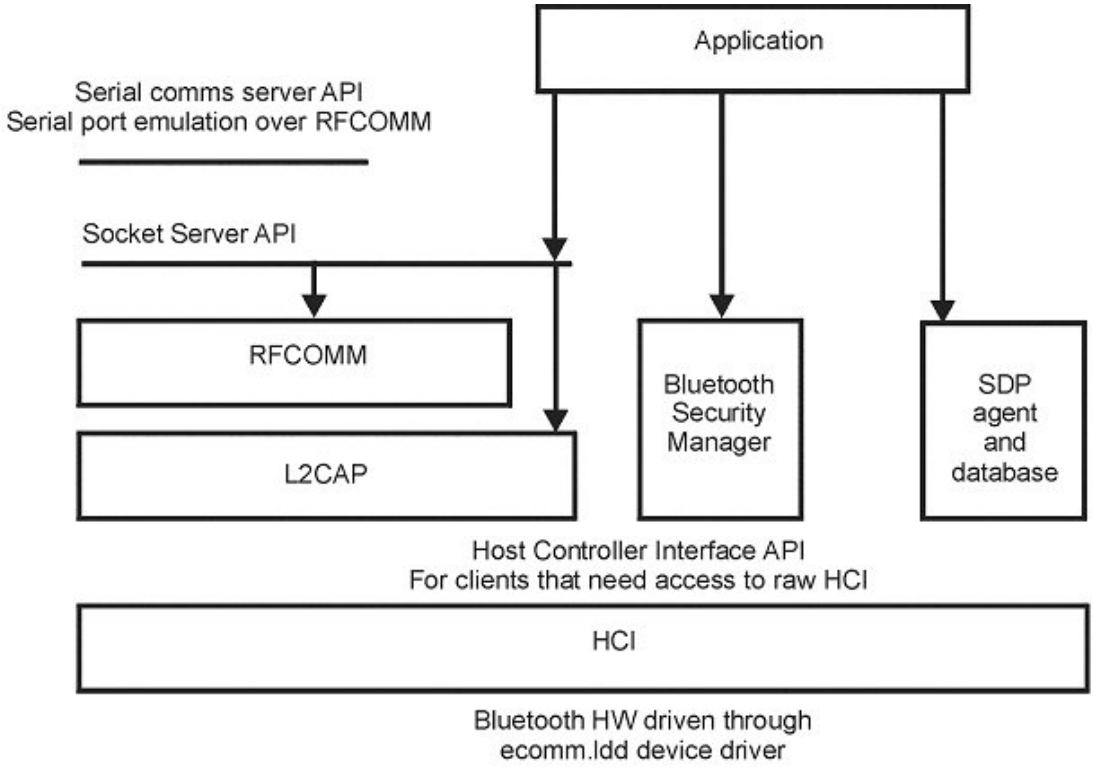


Figure 20: Bluetooth in Symbian OS

BT.PRT module contains also Bluetooth Manager and Service Discovery Protocol servers (Figure 20). The Bluetooth Manager abstracts e.g. all User Interface (UI) interactions according to BT. Bluetooth Security Manager enables Bluetooth services to set appropriate security requirements for incoming connections. Service Discovery Protocol server handles SDP queries and appropriate responses.

Symbian OS supports also Serial port emulation (Bluetooth Comms server module BTCOMM.CSY), which provides number of virtual serial ports for different services running over RFCOMM socket functionality. / 22 /

Next chapter concentrates on the practical part of the thesis. Requirements, design objectives and integration of Bluetooth component are presented there.

6. Software component for Bluetooth networking

6.1. Introduction to problem domain

The practical part of the thesis was to design and implement a reusable software component for Bluetooth networking. Data communications related issues have always been difficult to design and implement. Especially for typical software engineers implementing network features for their applications have always taken a lot of time and developing and testing efforts. Often these solutions are very problem domain specific and reusing them as such is quite difficult.

The software component that is developed offers a general API for networking and Bluetooth implementation to it. Later also Symbian OS socket based infrared implementation will be integrated to component. Integrating new bearers to the component makes finding common and variable features between different implementations far easier. This way it is possible to increase quality, performance and reusability of the component.

6.2. Component requirements

Bluetooth component was developed as an independent component project but a certain Digia's product development project acted as a client for this project. It set following requirements for the component:

Requirement #	**1**
Type	Functional
Name	**General API for bearer independent connectivity**
Description	Reusable general API for bearer independent connectivity. Through same generic interface the usage of different communications bearers (BT, IrDA etc) would be easier for application developer. Upper layer (easier to use interface) for Symbian OS socket facilities.

Rationale	An ordinary application developer isn't always specialist in developing communications software. Component and API enable faster and easier application development and the developer can concentrate more on developing the application logic.
Requirement #	**2**
Type	Functional
Name	**Network construction**
Description	Component handles all basic procedures needed for establishing a Bluetooth network connection (e.g. service registration, device inquiry, service discovery, connection establishment)
Rationale	Before data can be sent over BT, a connection must be established and it isn't always so easy procedure especially for inexperienced application developers.
Requirement #	**3**
Type	Functional
Name	**Multicasting support**
Description	Current BT Socket API of Symbian OS doesn't support multicasting (sending data to several other BT devices simultaneously), this feature will be implemented as simulated multicast, if the performance of current hardware is capable for it.
Rationale	For some applications only point-to-point Bluetooth connection would be a clear limitation for the functionality of application.
Requirement #	**4**
Type	Functional
Name	**Reading data, closing connections etc.**
Description	Readind data, closing connections, releasing reserved resources etc.
Rationale	Obvious features for component.
Requirement #	**5**
Type	Functional
Name	**Device Information Storage**
Description	Storage for saving information of devices in established Bluetooh piconet.
Rationale	Storage is needed for internal functionality of component and it may also offer some services for the user interface of application.
Requirement #	**6**
Type	Functional
Name	**Adding new members to BT network**
Description	Possibility to add new members to already established BT network.

Rationale	Desirable feature for the component because otherwise the whole piconet should be built again if the new Bluetooth device would want to join it.

Table 5: Main requirements for Bluetooth component

6.3. Architecture of component

Bluetooth component is so-called middleware component, this means that it uses operating system facilities and component is used by business-specific applications. Basically the component isn't visible directly to end-user. Only some operating system supported user interface notes (Bluetooth UI, e.g. Bluetooth device inquiry dialog) are shown to user because they are essential part of the functionality of the component. They provide also the correct look-and-feel that is needed in all application based on Series 60 Platform.

A high-level architecture of the component is shown in following UML diagram.

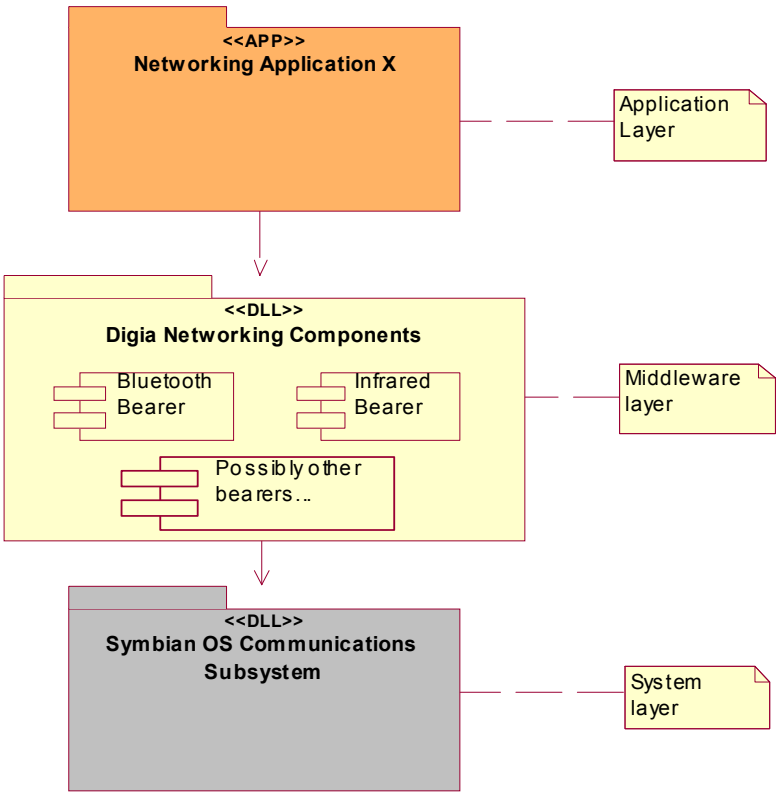


Figure 21: Architecture of Bluetooth component

6.4. Design of component

6.4.1. Class hierarchy

Following UML diagram shows the internal class hierarchy of the component. Only classes and their relationships are presented, methods and attributes would possibly make the diagram too unclear. All classes of BT component (not grey coloured system classes) and their functionality are described in Table 6.

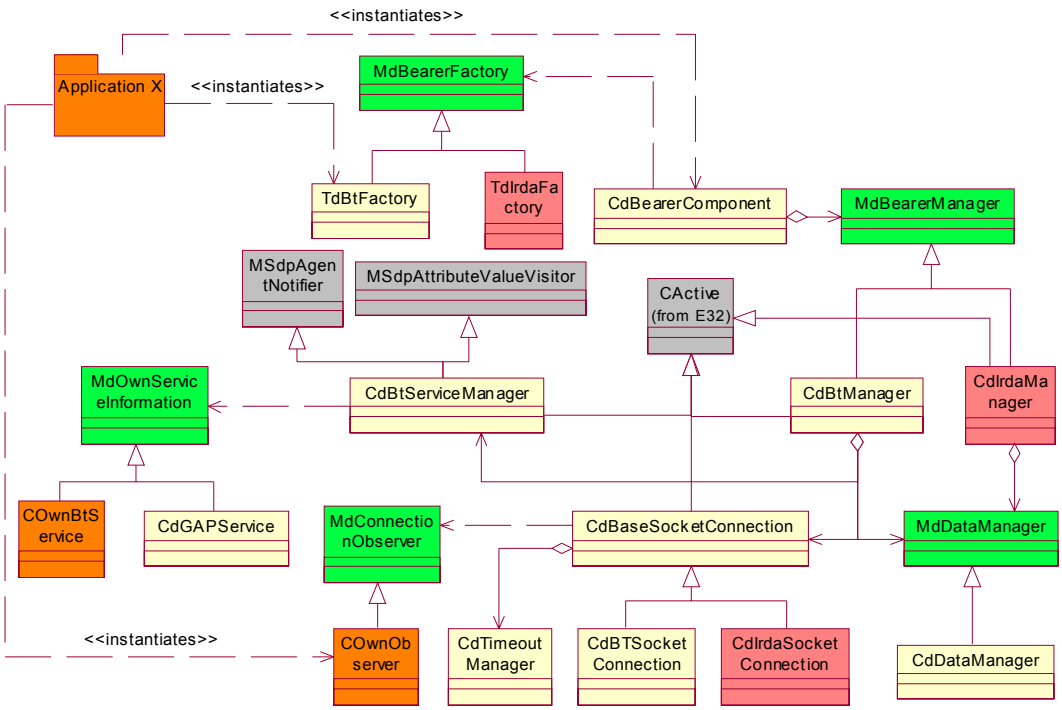


Figure 22: Class hierarchy for Networking API and Bluetooth implementation

Class	Description
`CdBearerComponent`	The main class of the component defines the interface for it (public header of API on appendix 2) and acts as a facade for internal functionality. Contains methods for finding Bluetooth services, establishing connections, sending and reading data etc.
`MdBearerFactory`	Interface for bearer factories. `CdBearerComponent` is instantiated with implementation of this class, but as `CdBearerComponent`

	knows only this interface, it doesn't need any changes as bearer is chosen at run-time on applications that use component.
`TdBtFactory`	Implements `MdBearerFactory` interface for Bluetooth bearer.
`TdIrdaFactory`	Implements `MdBearerFactory` interface for infrared bearer (integrated later).
`MdConnectionObserver`	Observer interface that is used for notifying the client application about occurred asynchronic events in component. The client of component implements this interface with its own client-specific class (=`COwnObserver`).
`MdBearerManager`	Interface for managing the basic functionality of different bearers (BT or infrared). `CdBearerComponent` uses concrete managers through this interface.
`CdBtManager`	Implementation for Bluetooth manager. This class is used for managing all basic BT functionality.
`CdIrdaManager`	Implementation for infrared manager (integrated later).
`MdDataManager`	Interface for data manager.
`CdDataManager`	Implementation for data manager, see chapter 6.5.3.
`CdBaseSocketConnection`	Class for encapsulating Symbian OS socket functionality. This class contains concrete sockets that are used for communications between devices. Base class for BT and IrDa socket classes, includes common functionality of both bearers.
`CdBTSocketConnection`	`CdBaseSocketConnection` derived class that implements Bluetooth specific connection establishment with Symbian OS sockets.
`CdIrdaSocketConnection`	`CdBaseSocketConnection` derived class that implements infrared specific connection establishment with Symbian OS sockets (integrated later).
`CdTimeoutManager`	Handles timeouts that may occur in connection establishment and sending / receiving data.
`CdBtServiceManager`	This class is storage for all Bluetooth service information. It is used for inquiring devices and it registers services and parses attributes of remote services. Concrete service definitions and attribute parsing are used through `MdOwnBtService` interface.
`MdOwnBtService`	Interface for own service definitions and attribute parsers.
`CdGAPService`	Internal implementation for service record. This is a minimal service record implementation and it fulfils the need of Bluetooth General

	Access Profile / 23 / In most cases, this implementation is adequate if there is no need to implement a profile application (e.g. file transfer, dial-up connection applications etc.) For example, a Bluetooth enabled game could use this implementation directly and the developer doesn't need to think Bluetooth service issues etc at all.
`CdOwnService`	If application needs to implement some BT profile or there is other need to add more service information to the service record, then the developer must implement his own service class. `MdOwnBtService` provides interface for it.

Table 6: Descriptions for classes of Bluetooth component

6.4.2. Commonality and variability analysis

SCV analysis (chapter 3.6) was quite easy to do for component. Initial requirements stated that component should offer generic API for different bearers. The client project had need for easy-to-use component for Bluetooth and infrared communication between two mobile devices. High-level SCV analysis is presented in following table:

Part of analysis	Analysis
Scope	Generic API and implementation for Bluetooth, later also infrared integration to API.
Commonality	Common features that are basis for generic API: establish connection, read data, send data, release resources etc.
Variability	Service handling is an essential part of Bluetooth but infrared doesn't need that kind of functionality at all.

Table 7: SCV analysis for Bluetooth component

6.4.3. Design patterns

Based on previous SCV analysis, component should offer a single class as an interface to component's features. This class contains common features that both bearer implementations need. Service handling was variable feature that is needed only with Bluetooth so there should be a way to attach it to the component without making use of infrared too complicated.

Following design patterns are utilized in the design of Bluetooth component: Facade, Factory and Adapter / 25 / . Classes that implement these patterns can be found from partial class hierarchies (Figure 23). Component becomes a black box, when a facade pattern is used. There is only a single entry point (class `CdBearerComponent`) to the component.

Factory pattern enables implementation of bearer independent interface for component. Adapter (described in chapter 3.8) is used for adapting Bluetooth service handling and possible own service implementations to `CdBtManager` class inside the component.

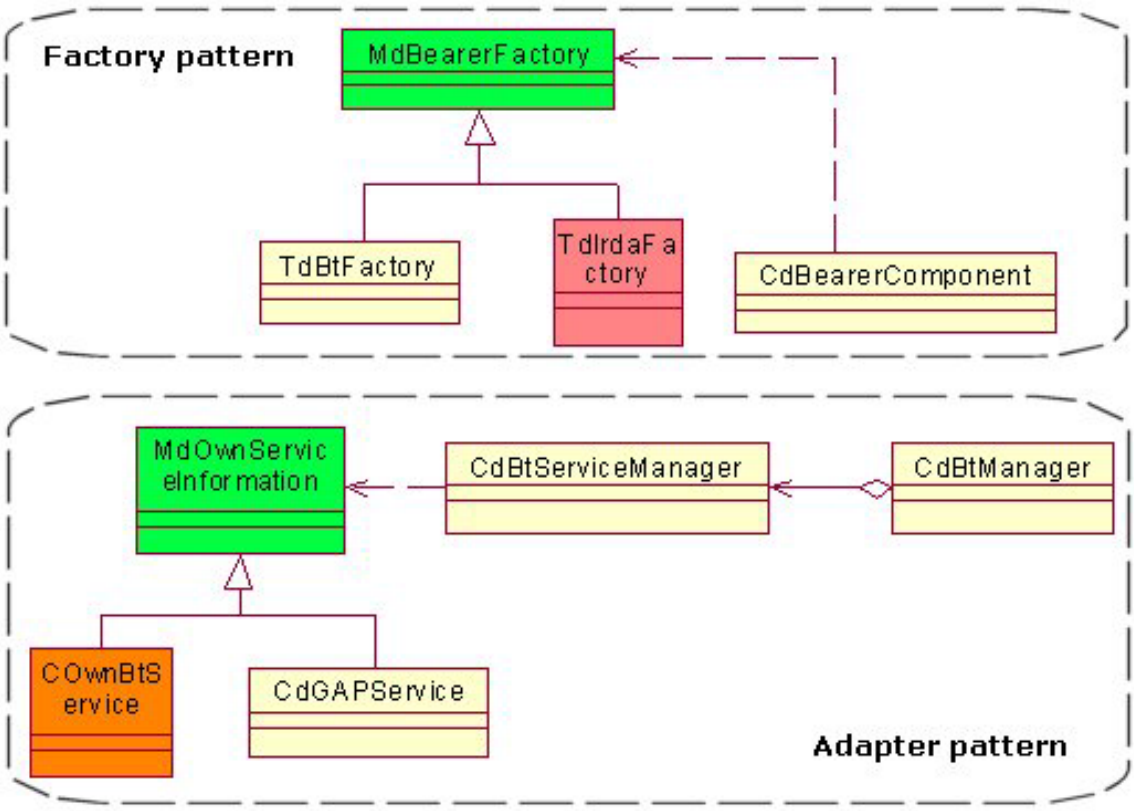


Figure 23: Examples of Bluetooth component's design patterns

6.5. Component implementation

6.5.1. Development environment

Bluetooth component was developed with C++ programming language for Symbian OS and especially Series 60 Platform (chapter 2.3.1) but there are no limitations to port component to other reference designs. Symbian OS Bluetooth architecture provides Bluetooth UI, which is used for inquiring devices and setting some security settings. This UI is available on all reference designs that are based on Symbian OS GT v6.1 or newer (Series 60 or UIQ).

6.5.2. Connection establishment procedure

Instantiation process of component is described in appendix 3. Figure 24 shows how point-to-point Bluetooth connection is established between two BT devices.

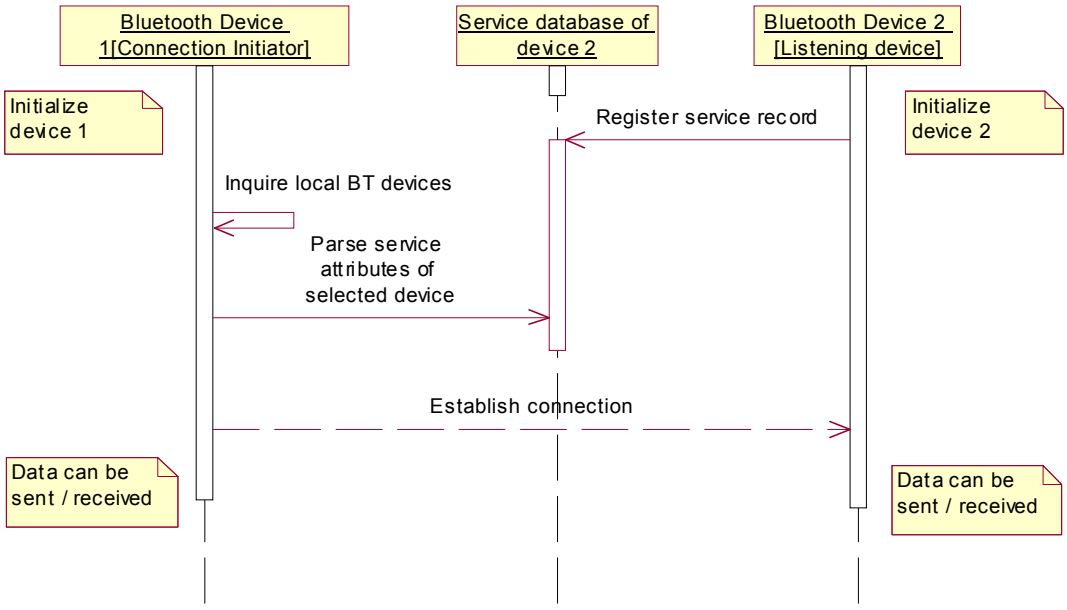


Figure 24: Connection establishment procedure

6.5.3. Datamanager

Datamanager is used for handling data packets (Figure 25) that are transmitted over communication link. Datamanager adds control information to data before it is sent. This information may contain control code (e.g. for detecting different kind of packets) and information about the actual length of data, because otherwise the remote device doesn't know how much data there will be coming over data link. When a communication socket on the remote device has read the data, datamanager will examine the header part of it. Length information can be used for allocating correct amount of memory for incoming data buffer.

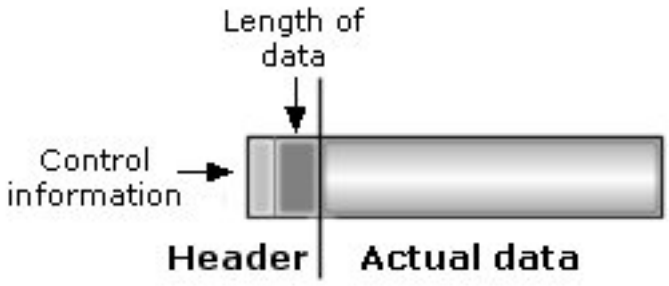


Figure 25: Example of data packet

6.5.4. Metrics of the component

The main objectives for software metrics are:

- To better understand the quality of the software product
- To assess the effectiveness of the process
- To improve the quality of work performed at a project level / 29 /

Only few metrics of software are absolute. Absolute metrics (e.g. lines of source code, size of binary file, etc) have certain clear values, an ordinary software developer can conclude easily from them which value is better than the other. For example, a binary size of application A is 54 kB and size of application B is 76 kB, it is quite obvious which is better value in metrics sense.

Most of the software metrics are derived (=value of the metrics must be calculated, it isn't visible directly from software) and the values are relative. Comparing relative values is much more difficult that comparing absolute ones. Often the result of comparison isn't so clear as with absolute metrics. If a relative value of some metric is "5" for some application and "6" for some other, it is quite hard to conclude which value is better. Realistic conclusions can be done only when the differences of values are significant, for instance the other value is "3" and the other one is "174".

Following table contains the main technical metrics of the component:

Metrics:	Value:	Description:
Lines of source code (LOC)	**≈ 2000**	*[Absolute metric]* Often the most familiar measure for code. Actually not very good measure for object-oriented technology, but gives information about the magnitude of developed software module.
Binary size	**11 kB**	*[Absolute metric]* On disk footprint. Size of release version of component DLL build for THUMB processor environment / 28 / [BT component: Nokia 7650 smartphone]
Memory allocation	**< 8 kB**	*[Absolute metric]* Run-time footprint. Memory consumption when component is used in real device environment.
Resource usage strategy	**Linear-deterministic**	*[Absolute metric]* How resources (memory, files etc.) are used, when they are allocated, released etc. *Static* = Resources are allocated in the beginning of component execution and there are no dynamic reservation later on *Linear-deterministic* = Most of the resources are allocated in the beginning, least important resources can be allocated dynamically later

		Random = Resources are allocated whenever they are needed
Granularity	**7 / 9** **= 0,8**	*[Relative metric]* Functionalities / classes (/LOC) (=cohesion). [BT component: service methods in public API / concrete classes]
Abstractness	**5 / 9** **= 0,6**	*[Relative metric]* Interfaces / concrete classes.

Table 8: Some technical metrics of Bluetooth component

6.6. Component integration

6.6.1. Reusing component

Reusing component in other application is quite straightforward. Here is a short procedure how to use it:

1. Developer uses directly Bluetooth component's dynamic link library file (DLL) in his own application. Only header files of exported classes are needed for compiling and LIB file is used for linking modules of application.
2. In source code level: `CdBearerComponent` object is created with correct bearer factory object (=`TdBtFactory`) and observer object (=`COwnObserver`):

```
// How to use Bluetooth component
// © Digia 2002

// We use BT factory class to instantiate Bluetooth component

TdBtFactory btFactory;

// Own observer class that implements MdConnectionObserver interface
// This is used for notifying client application about asynchronic
// events that have occurred in component

iObserver = COwnObserver::NewL();

// Component is instantiated with concerete factory and observer
// objects, also timeouts for connection establishment and sending/receiving
// data are initialised (values are microseconds)

iBearerComponent = CdBearerComponent::NewL( btFactory, iObserver, 30000000, 5000000
);
```

```
// Bearer component is initialised with service that
// is identified with ID: KServiceId

iBearerComponent->InitL( KServiceUid );

// Component is now ready for using in application
// → device inquiry, service discovery,
// connection establishment, sending data, etc.
```

6.6.2. Testing application

Testing application of the component is just a simple user interface that is used for connecting to other device and sending data to it. Couple of screen captures are presented in Figure 26.

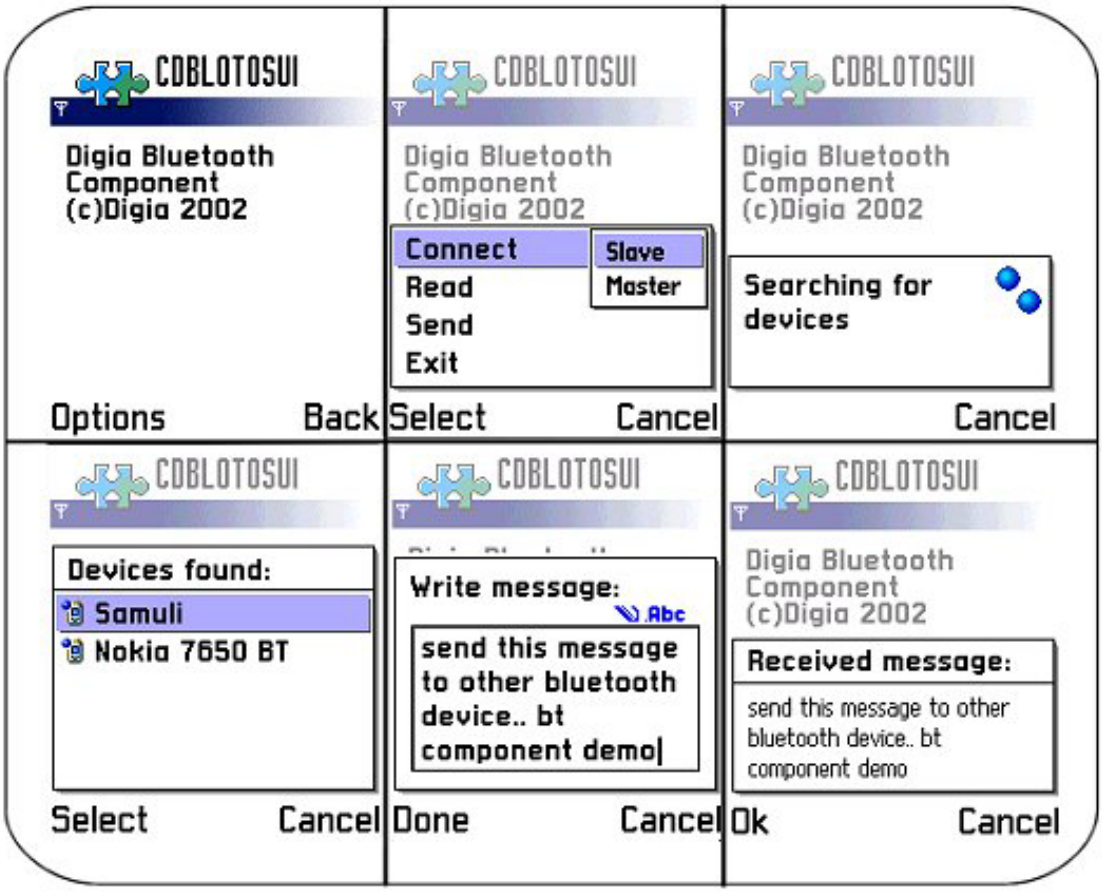


Figure 26: Screen captures of testing application

6.7. Future development

The current version of component is just the first delivery of it. Component provides networking API and point-to-point Bluetooth implementation. Establishing BT connections and reading or sending data is quite straightforward with it.

However, the development of component is not finished yet. In near future infrared implementation is integrated to it and some new features may be implemented. Component needs systematic functional testing and design and code inspections. In addition, detailed documentation hasn't been written yet.

Integration to other application, which is the most important part, hasn't been done yet. The developer of component may think that some features are obvious and the component is "easy-to-use". Other developers that aren't familiar with domain (=Bluetooth in this case) may find component difficult or illogical to use. Therefore, feedback from different developers and several integration rounds are needed that component could become mature enough. Without proper feedback, the component may fit just for one purpose and the real reusability remains as a daydream.

Component will be integrated at least twice in exercise projects before it is used in Digia's product development project, which has acted as a client for the component. As defined in Digia's reuse process (chapter 4.4.4), the component will be stored to component library. Component library contains always the newest version of particular component and all material related to it (source code, documentation, possibly binary files etc).

7. Conclusions

The purpose of this thesis was to present different aspects of software reuse. What does it really mean and benefits and obstacles of reuse from technical and nontechnical point of views. Symbian Operating System and wireless Bluetooth radio technology was also briefly introduced. The practical part of thesis concentrated on developing reusable Bluetooth component for Symbian OS.

Dictionary definition for reuse is quite evident: “To use again, especially after reclaiming and reprocessing”. Reuse in software engineering is an area that has been researched a lot during last few decades. Nevertheless, many software development organizations have faced problems when they have tried to establish reuse processes for their software development.

The benefits of reuse are apparent and significant. The quality and performance of software improves, resources can be allocated more efficiently in projects and the development-cycle of software products can be significantly shortened. Software reuse has a positive effect for all fundamentals of software projects: efforts, costs and time.

The biggest obstacles of software reuse are demanding and urgent projects, attitudes, insufficient support from upper management of software organization and a lack of training of developers and managers. That is, the reason for inefficient or nonexistent reuse in development organization is often nontechnical. For correcting previous issues, the reusability must become an essential part of daily work. Reuse program, processes and adequate organization are needed and sufficient resources must be available for them. Appropriate training and support from management are essentials for successful reuse program.

Software components are important part of software reuse. Importance of adequate knowledge of development domain and object-oriented design principles and patterns must

be considered during component development. Without these fundamentals, the developing of reusable component could turn to development of component that fits only for one purpose.

8. References

All product images (Symbian, Nokia, Sony-Ericsson) presented in this thesis are free for non-commercial use. Images are available on following web site: http://www.symbian.com/press-office/picture-library.html [referenced 13.8.2002]

/ 1 / Digia: Digia Software Process, Digia 2001, [Confidential] available: Digia Intranet

/ 2 / Symbian: Symbian OS v7.0 [Internet], available: http://www.symbian.com/technology/symbos-v7x-det.html [referenced 13.8.2002]

/ 3 / Open Mobile Alliance [Internet], available: http://www.openmobilealliance.org/ [referenced 13.8.2002]

/ 4 / Nokia: Forum Nokia [Internet], available (registration is needed): http://forum.nokia.com/ [referenced 13.8.2002]

/ 5 / UIQ user interface for Symbian OS v.7.0 [Internet], available: http://www.symbian.com/technology/UI/uiq.html [referenced 13.8.2002]

/ 6 / Digia: Programming for the Series 60 Platform and Symbian OS, John Wiley and Sons Ltd, 2002

/ 7 / Martin Tasker et al: Professional Symbian Programming, Wrox Press Ltd, 2000

/ 8 / J. Jipping: Symbian OS Communications Programming, John Wiley and Sons Ltd, 2002

/ 9 / Symbian OS Communications - Design [Internet], available: http://www.symbian.com/developer/techlib/papers/comms-des/comms-des.htm [referenced 13.8.2002]

/ 10 / Symbian OS Communications – Implementation [Internet], available: http://www.symbian.com/developer/techlib/papers/comms-imp/comms-imp.htm [referenced 13.8.2002]

/ 11 / Ivar Jacobson, Martin Griss and Patrik Jonsson: Software Reuse: Architecture Process and Organization for Business Success, Addison-Wesley Co, 1997

/ 12 / Johannes Sametinger: Software Engineering with Reusable Components, Springer, 1997
/ 13 / Eila Niemelä, Seppo Kuikka et al.: Teolliset komponenttiohjelmistot, TEKES Teknologiakatsaus 89/2000
/ 14 / Alan Shalloway and James R. Trott: Design Patterns Explained: A New Perspective on Object-Oriented Design, Addison-Wesley Co, 2001
/ 15 / Sun: Enterprise JavaBeans - EJB, [Internet], available: http://java.sun.com/products/ejb/ [referenced 13.8.2002]
/ 16 / Microsoft: Component Object Model – COM, [Internet], available: http://www.microsoft.com/com/ [referenced 13.8.2002]
/ 17 / Object Management Group: Common Object Request Broker Architecture - CORBA, [Internet], available: http://www.omg.org/gettingstarted/corbafaq.htm [referenced 13.8.2002]
/ 18 / Symbian: Symbian OS Developer Library: Symbian OS v7.0: ECOM, [Internet], available: http://www.symbian.com/developer/techlib/sdl.html [referenced 13.8.2002]
/ 19 / James Coplien, Daniel Hoffman and David Weiss: Commonality and Variability in Software Engineering, IEEE Software 15(6), November, 1998, [Internet], available: http://www1.bell-labs.com/user/cope/Mpd/IeeeNov1998/ [referenced 13.8.2002]
/ 20 / Jennifer Bray and Charles F. Sturman: Bluetooth – Connect Without Cables, Prentice-Hall Inc, 2001
/ 21 / David Kammer, Gordon McNutt, Brian Senese and Jennifer Bray: Bluetooth – Application Developer's Guide, Syngress Publishing Inc, 2002
/ 22 / Bluetooth and Symbian OS [Internet], available: http://www.symbian.com/technology/standard-blue.html [referenced 13.8.2002]
/ 23 / Bluetooth Specifications [Internet], available: http://www.bluetooth.com/dev/specifications.asp [referenced 13.8.2002]
/ 24 / Symbian: Symbian ownership, [Internet], available: http://www.symbian.com/about/ownership.html [referenced 13.8.2002]
/ 25 / Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides: Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

/ 26 / Robert C. Martin: Design Principles and Design Patterns, [Internet], available: http://www.objectmentor.com/resources/articles/Principles_and_Patterns.PDF [referenced 13.8.2002]
/ 27 / Hans-Erik Eriksson and Magnus Penker: UML, IT-Press, 2000
/ 28 / ARM Ltd, Thumb – an extension to 32-bit ARM architecture, [Internet], available: http://www.arm.com/armtech/Thumb?OpenDocument [referenced 13.8.2002]
/ 29 / Roger S. Pressman: Software Engineering – A Practitioner's Approach, The McGraw-Hill Companies Inc, 1997
/ 30 / Prosessori-Uutiset, "Älyrenkaaseen 433 MHz piirit", 19.10.2001, [Internet], available: http://www.prosessori.fi/uutiset/uutinen.asp?id=41405 [referenced 13.8.2002]
/ 31 / Nokia: Series 60 Platform 1.0 – Product Overview, [Internet], available: http://www.nokia.com/networks/systems_and_solutions/files/content/mim_series_60_version10_po.pdf [referenced 13.8.2002]

Appendix 1: UML notation

Figure 27 shows notation for UML packages and components. Packages are used for organizing semantically similar elements to groups / 27 /

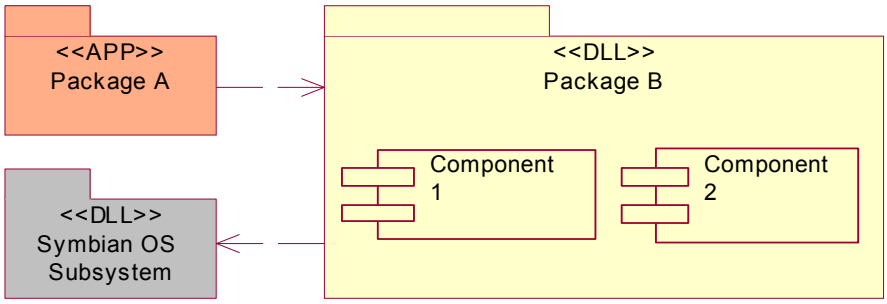


Figure 27: UML packages

Six classes and their relationships are presented in Figure 28. Roles of relationships are shown as stereotypes, e.g. “inherits” or “delegates”. Normal classes are coloured with yellow, abstract interface classes (so called M-classes in Symbian OS) are green and system classes are grey. External packages or classes are coloured with orange colour.

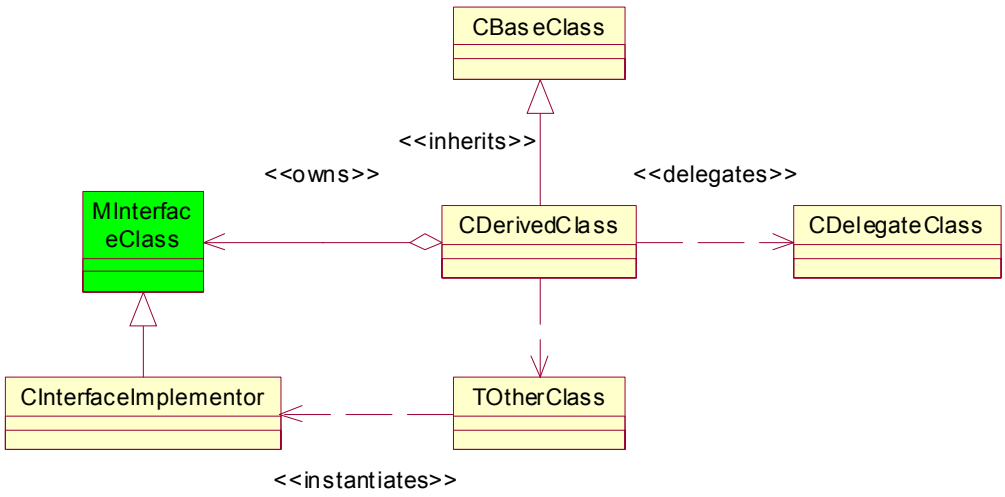


Figure 28: Class hierarchies in UML notation

A single class and its attributes and operations are shown in Figure 29; also a class declaration in C++ programming language is presented after it.

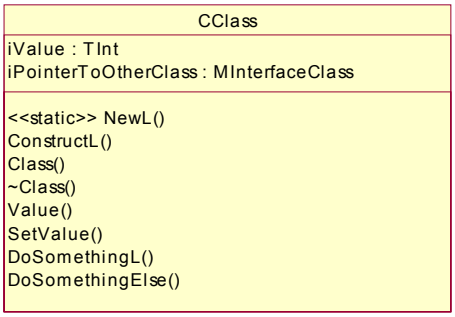


Figure 29: Symbian OS C-class and its attributes and operations

```
// ============================================================================
// Example of C++ class and its attributes and operations
// ============================================================================

class CClass : public CBase
        {

        public: // Constructors

                static CClass* NewL( MInterfaceClass* aPointer );
                ~CClass();

        private: // Constructors

                void ConstructL( MInterfaceClass* aPointer );
                CClass();

        public: // New methods

                TInt Value();
                void SetValue( Tint aValue );

                void DoSomethingL();
                void DoSomethingElse();

        private: // Data

                TInt iValue;
                MInterfaceClass* iPointerToOtherClass;
        };

  // ============================================================================
```

Sequence diagrams (Figure 30) are used for modelling the dynamic behaviour of the system. Sequence diagram shows clearly interaction and messages between different objects in same order as they will occur in real system.

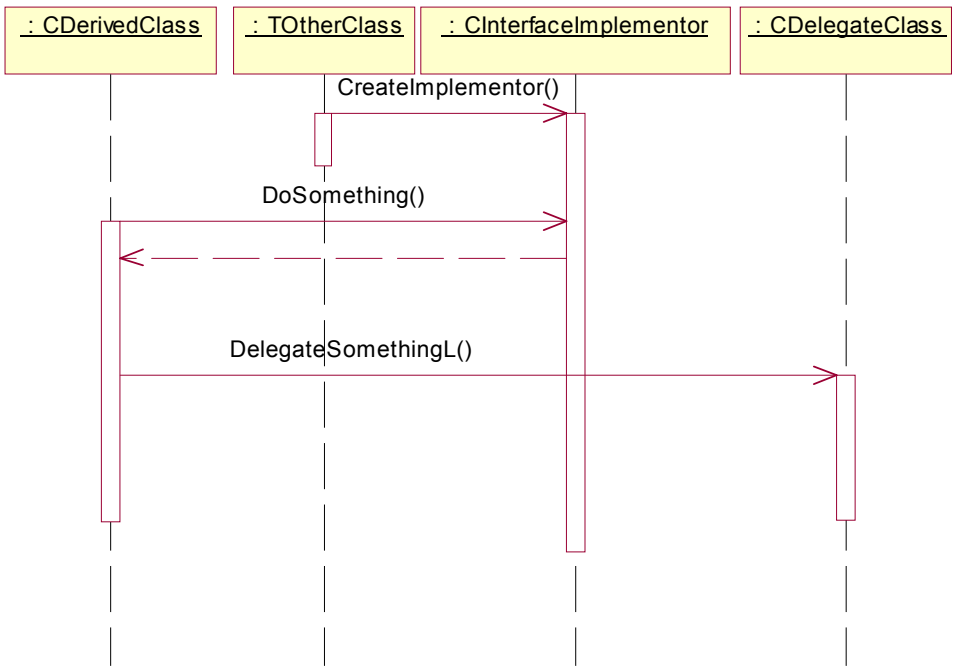


Figure 30: Sequence diagram

Appendix 2: Public API of BT component

Here is the public API of Bluetooth component:

```
// ============================================================================
// CdBearerComponent, public API for Digia Bluetooth Component
// © Digia 2002 [version 0.09, 28.8.2002]
// ============================================================================
class CdBearerComponent : public CBase
  {
  public:
          //   Static constructor
          IMPORT_C static CdBearerComponent* NewL( MdBearerFactory& aFactory,
MdConnectionObserver* aObserver, TTimeIntervalMicroSeconds32 aConnectionTimeout,
TTimeIntervalMicroSeconds32 aDataTimeout );

          //   Destructor
          IMPORT_C virtual ~CdBearerComponent();

          // Initialize BT component with GAP service or own BT service implementation
          IMPORT_C void InitL( const TInt aServiceUuid );
          IMPORT_C void InitL( MdOwnServiceInformation* aServiceInformation, const
TInt aServiceUuid );

          // Connect, synchronous and asynchronous versions
          IMPORT_C TInt ConnectL( TBool aMaster, TBool aThisIsSynchronous = EFalse );

          // Close connection
          IMPORT_C void Close();

          // Inquires other BT devices and discovers service that
          // is defined with aServiceUuid parameter
          IMPORT_C TInt FindServiceL( const TInt aServiceUuid );

          // Read data to descriptor, synchronous/asynchronous and 8/16 bit versions
          IMPORT_C TInt ReadL( TDes8& aData, TBool aThisIsSynchronous = EFalse );
          IMPORT_C TInt ReadL( TDes16& aData, TBool aThisIsSynchronous = EFalse );

          // Read data, synchronous/asynchronous and 8/16 bit versions
          IMPORT_C TInt SendToL( const TDesC8& aData, TBool aThisIsSynchronous =
EFalse );
          IMPORT_C TInt SendToL( const TDesC16& aData, TBool aThisIsSynchronous =
EFalse );

          // Returns the lentgh of received data
          IMPORT_C void LengthOfDataL( TInt& aLength );

  private:

          //   C++ constructor
          CdBearerComponent();

          //   Second phase constructor
          void ConstructL( MdBearerFactory& aFactory, MdConnectionObserver* aObserver,
TTimeIntervalMicroSeconds32 aConnectionTimeout, TTimeIntervalMicroSeconds32
aDataTimeout );

  private: // data

          MdBearerManager* iBearerManager;
          MdOwnServiceInformation* iServiceInformation;
          TBool iGapService;
          TPtr8 iDataPtr;
   };
```

Appendix 3: Instantiation of component

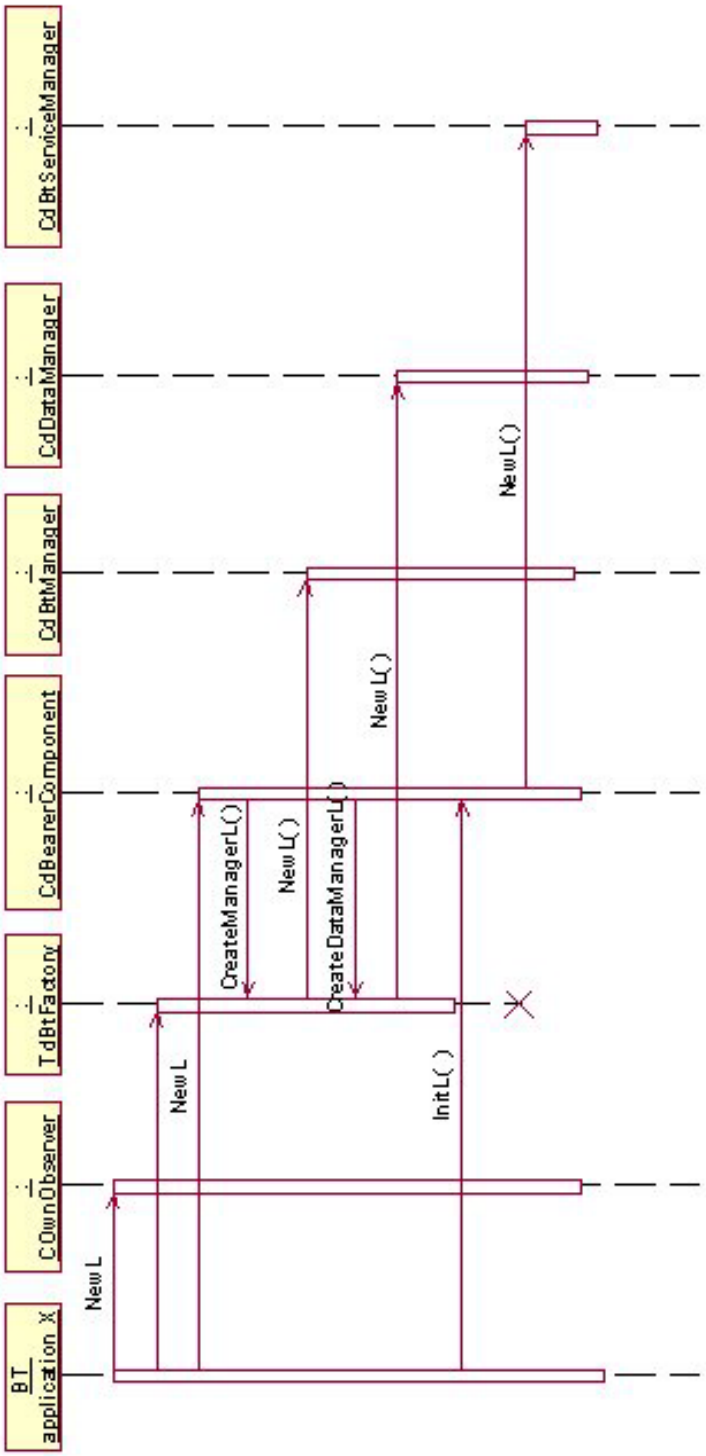


Figure 31: Instantiation of Bluetooth component