Diploma Thesis - Spring Term 2007/2008

Wireless Ad Hoc Podcasting on Symbian

Ákos Bakos
akosb@tik.ee.ethz.ch

22nd May, 2008

Internal Supervisors:	Dr. Vincent Lenders, ETH Zurich
	Bernhard Distl, ETH Zurich
External Supervisor:	Dr. Károly Farkas,
	University of West Hungary
Host:	Prof. Dr. Bernhard Plattner,
	ETH Zurich

Abstract

Podcasting has become popular for dissemination of content over the Internet. It is based on subscriptions where software clients query servers for updates of subscribed content feeds. The main limitation with the current system is the inflexible separation of downloading to a docked media player and expending of the data when on the move. The solution proposed here uses peer-to-peer synchronization to exchange content directly between neighboring devices.

The goal of this thesis is to develop a prototype content distribution application for Symbian phones which makes use of their ad hoc capabilities (Wifi) in order to directly connect them. The application should allow users to exchange pictures and videos that were generated on the mobile phones.

Thus, the existing podcasting code has been designed and implemented in C++ from an earlier project that we have ported to Symbian by adapting the platform independence library. We have developed a GUI in Symbian C++ that communicates with the core application. Our programs was primarily designed for Nokia N93 and N95 handhelds.

Kivonat

A podcasting lehetővé teszi multimédia anyagok közzétételét az interneten, úgy, hogy a felhasználók feliratkozhatnak az adott anyag újdonságait tartalmazó *feed*re. A podcasting online tartalmak terjesztésének egy újszerű formája a feliratkozásos modell miatt, amely egy *feed* segítségével tájékoztatja a felhasználót az új állományról.

Ennek a rendszernek a fő hátránya, hogy mozgás közben új tartalmakat fogadjunk és küldjünk. Erre a problémára megoldást jelent a közvetlen szinkronizáció, amely a közvetlen szomszédok között cseréli ki a tartalmakat.

Ennek a diplomamunkának a célja, egy olyan adatmegosztó alkalmazás fejlesztése Symbian platformra, amely közvetlenül kommunikál a telefonok interfészein keresztül (bwifi). A felhasználók ezzel a programmal a telefon által készített képeket és videókat tudják kicserélni egymás között.

A használt podcasting programot C++ programnyelven implementáltak egy korábbi projekt során, amely a podcasting elvét kiterjeszti a mobil felhasználókra. Ezt a podcasting alkalmazást fordítottunk le Symbian platformra, platformfüggetlen könyvtárak segítségével. Továbbá egy Symbian C++ programnyelven megírt grafikus felhasználói felületet fejlesztettünk, ami a központi alkalmazással kommunikál. A program Nokia N93-as és N95-ös típusú mobil telefonokon működik.

Acknowledgements

During the work on this thesis a number of people supported me in their own ways and I want to express my sincere gratitude to everybody who made this thesis possible, especially to:

- *Prof. Dr. Bernhard Plattner*, for giving me the opportunity to write my master thesis at the Communication Systems Group;
- *Károly Farkas*, for the supervision of my work.;
- *Dr. Vincent Lenders*, for the great guidance and support during the whole project;
- *Bernhard Distl*, for interesting discussions and provided me with new ideas;
- *My parents*, for the generous support during my whole diploma thesis writing;
- *Réka*, for her inspiring ideas.

Zurich, May 22th 2008

Ákos Bakos

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Wireless Ad Hoc Podcasting

There are two modes of public content distribution that are commonly used for mobile devices today. One is podcasting, where the device downloads contents when docked to a computer with an internet connection and the other is live streaming that is available in 3G cellular networks. Podcasting is enabled by the massive storage that is available in mobile device. The distribution is limited to the contents that are available at the time of the downloading and contents published thereafter remain inaccessible. Even further increases in storage capacity will not alleviate this. Wireless internet access, via public IEEE 802.11 networks for instance, may provide more frequent downloading opportunities when the devices are on the move. The second alternative are the 3G systems. They provide good coverage and thereby also continuous access to contents and do not require much storage on the device. The technological limit is the capacity in a cell, around 5 Mb/s, that quickly may become saturated if streaming becomes popular. Another concern is that operators might limit the available contents to those that they or their affiliated partners, provide (the so called wireless net neutrality issue).

The system, PodNet, is based on WLAN mode and aims at increasing the opportunities for devices to obtain contents while being mobile. PodNet utilizes connections with access points, when in range and it distributes contents opportunitistically from node to node otherwise. This opportunistic node-to-node distribution is important for extending the availability of contents beyond the reach of the infrastructure and it enables distribution of contents that are generated by the mobile nodes without a supporting infrastructure. It is possible to combine PodNet and 3G distribution so that contents are primarily received to a mobile device by one of the two systems. The other serves then as a back up in case of unavalibility of contents through the primary system.

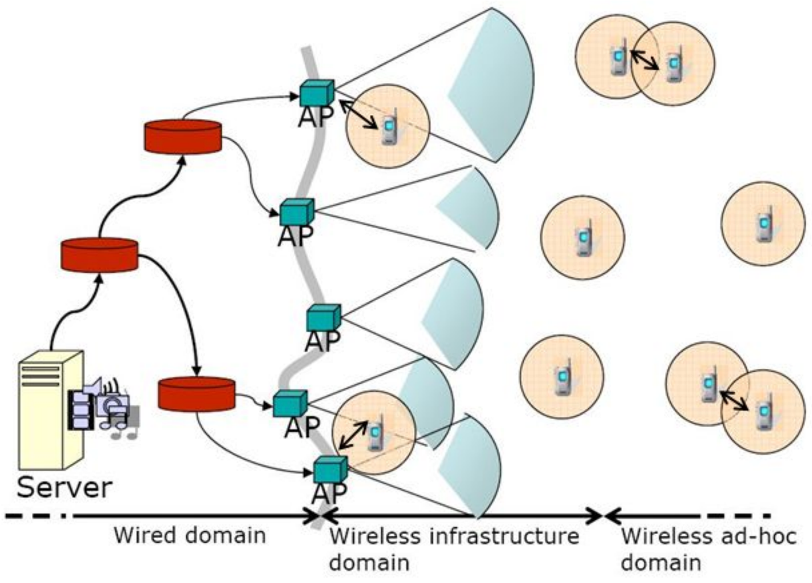


Figure 1.1. Contents are provided by means of podcast via gateways and they are redistributed in the ad hoc domain.

This work describes how the podcasting concept can be expanded for public (i.e., open and unrestricted) peer-to-peer delivery of contents amongst mobile nodes. Contents are provided in a wireless broadcasting area either by access points or by mobile nodes which have been filled with contents off line (Figure 1.1). In the first case, each access point fetches contents from podcast servers across the Internet and forwards them to mobile nodes within its range. In the second case, a mobile node that has contents to share might provide data to another mobile node when they pass within radio range of one another. This is the new content distribution mode that we add to the existing one. Our ad hoc podcasting mode brings the following advantages. First, it provides nodes with contents when they are not connected to the Internet; second, it provides a new ad hoc broadcasting domain when also the sources of the data are mobile nodes (could be pictures or voice recordings from a mobile phone for instance). We will also refer to this wireless ad hoc mode of broadcasting as podcast in the hope of broadening the current concept (motivated by the use of the feet of a pod of users for content distribution).

Wireless ad hoc podcasting, as we present in this work, is an application based on the *delaytolerant broadcasting* concept that has been proposed in an earlier paper [1]. The sharing of contents is based on a solicitation protocol by which a node asks a peer node for content. Hence, there is no flooding of contents in the broadcasting area. Contents are organized into *channels*, and nodes solicit *episodes* for one or more channels.

The concept of channels allows for a higher hit rate of the queries than if they were for individual episodes of contents. The episodes of a particular channel will however reach a node in arbitrary order, and not all of them will be received; it is therefore important that all contents are provided in atomic units, which are short enough to be downloaded in a contact and which may be replayed without relation to other units. We believe that the podcasting application fulfills these requirements, since a channel could be composed of a mixture of entries of music, news items, weather updates and commentaries, such as the mix broadcast by many radio stations.

1.1.1 Thesis Contribution

This thesis is part of a larger research project (PodNet [3]) with the long-term goal to develop a delay-tolerant network architecture for content distribution on mobile phones. In this thesis we focus on implementation for Symbian platform an ad hoc wireless networking protocol for opportunistic exchange of podcasts between the mobile devices. We have read and analyzed the C++ ad hoc podcasting developed code in the Master thesis work of Clements Wacha [4] then we have created the GUI on Symbian platform.

1.2 Thesis Structure

The rest of this document is structured as follows:

- This chapter provides an introduction to this thesis.
- Traditional podcasting, as existing in the Internet and other concepts used in this work are described in the next chapter.
- Chapter 3 gives an introduction to the Symbian OS.
- Chapter 4 reviews briefly the Nokia platforms and gives an overview about the Series 60 3^{rd} edition.
- Chapter 5 describes our current implementation of this concept as a Symbian C++ application for Nokia N95 and Nokia N93 handhelds.

- Chapter 6 lists the hardware and software used for the implementation and contains the testing process.
- And Chapter 7 gives summary and conclusions.

Chapter 2

Related Work

2.1 Podcast

From Wikipedia [11]:

> A podcast is a series of digital-media files which are distributed over the Internet using syndication feeds for playback on portable media players and computers. The term podcast, like broadcast, can refer either to the series of content itself or to the method by which it is syndicated; the latter is also called podcasting. The host or author of a podcast is often called a podcaster.
>
> The term is a portmanteau of the words iPod and broadcast, the Apple iPod being the brand of portable media player for which the first podcasting scripts were developed (see history of podcasting). These scripts allowed podcasts to be automatically transferred to a mobile device after they are downloaded.
>
> Though podcasters' web sites may also offer direct download or streaming of their content, a podcast is distinguished from other digital media formats by its ability to be syndicated, subscribed to, and downloaded automatically when new content is added, using an aggregator or feed reader capable of reading feed formats such as RSS or Atom.

From a technical point of view a podcast is very similar to a regular news feed in RSS or Atom format. Both news feed formats divide content into channels (i.e. podcasts or feeds) and episodes (i.e. news items, entries). An episode may contain an enclosure (i.e. attachment). When talking about podcasts the enclosure is usually an MP3-file or another sort of audio file.

A podcast with one episode therefore consists of two files. The MP3-file and the XML file containing the Atom or RSS meta information. This concept gets very interesting if we use a podcasting application that is able to check the meta file automatically if new episodes have arrived. iTunes is one of those applications and it is also able to automatically download new episodes.

As podcasts as well as news feeds have become very popular the idea arises to extend their usage to wearable devices such as Nokia N93 and Nokia N95 handhelds. Unfortunately these devices are not always connected to the internet which prevents them from periodically updating their content. This in turn leads to our goal of creating an application that is able to fetch new episodes not only from the internet but also from other handhelds that happen to be in WLAN range.

2.2 Podnet

PodNet is a research project with the goal to build a communication infrastructure out of human-carried personal devices equipped with wireless capabilities like WLAN or Bluetooth. The portable devices include media players, digital cameras, PDAs, mobile phones and game consoles.

Inspired by social networking, we conceive a communication network which mimics how people spread information: data is exchanged between two devices based on solicitations during contacts, i.e., whenever two persons come into wireless range.

The contents in the system are made available either from the Internet or in the mobile domain (Figure 1.1). The contents are provided as channels, where each channel gathers contents for a specific topic. A mobile user that is interested in one or more particular topics subscibes to the corresponding channels on his mobile device. This subscription has only a local scope: it simply informs the user's device which channels it should retrieve contents for and does not involve any remote logging of the subscription. As we will see later in this section, channels are identified with a globally unique channel identifier. Therefore, a user subscribes to the channel by specifying the desired channel identifier. A user might learn about available channels and their identities when connected to the Internet, for example through repositories or search engines like iTunes, Google or Yahoo. However, PodNet also provides it own channel discovery mechanism. Users can subscribe to a well known discovery channel that includes meta information as well the identifiers of the available channels in the network. This channel is handled as a regular channel and it thus updated as nodes peer with each other on the move. After a user has subscribed to a list of channels, his or her device is ready to download contents. Every time the device has a connection to another mobile node, it will try to download episodes from the subscribed channel. A user may typically want to define some local policies. For example, if subscribed to many different channels, he or she might want to assign channel preferences to ensure that the favorite channels are downloaded first during the transfer opportunities. Another policy could control what specific items of contents are downloaded from an individual channel. The user might for example be intrested only in the latest episode on a news channel, whereas files from music channels are more independent of time and do not have to be updated on a regular basic. Also, users should be able to specify contents that are not of interest within a channel. If a user has for example already downloaded, consumed and deleted content, the device should obviously not try to download the same content again in the future.

Chapter 3

Symbian OS overview

Symbian OS is a proprietary operating system, designed for mobile devices, with associated libraries, user interface frameworks and reference implementations of common tools, produced by Symbian Ltd. It is a descendant of Psion's EPOC [1] and runs exclusively on ARM [2] processors [5].

3.1 Design

Symbian OS is characterized by: [6]

- **Integrated multimode mobile telephony:** Symbian OS integrates the power of computing with mobile telephony, bringing advanced data services to the mass marke;
- **Open application environment:** Symbian OS enables mobile phones to be a platform for deployment of applications and services (programs content) developed in a wide range of languages and content formats;
- **Open standards and interoperability:** with a flexible and modular implementation, Symbian OS provides a core set of application programming interfaces and technologies that is shared by all Symbian OS phones. Key industry standards are supported;

[1]EPOC is a family of operating systems developed by Psion for portable devices, primarily PDA's. Epoc is rumoured to derive from "Electonic Piece Of Cheese", or from epoch - the beginning of an era.

[2]The ARM architecture (previously, the Advanced RISC Machine and prior to that Acorn RISC Machine) is a 32-bit RISC processor architecture developed by ARM Limited that is widely used in a number of embedded designs.

- **Multi-tasking:** Symbian OS is based on a micro kernel architecture and implements full multi-tasking and threading. System services such as telephony, networking middleware and application engines all run in their own processes;
- **Fully object-oriented and component based:** the operating system has been designed from he ground up with mobile devices in mind, using advanced object oriented techniques, leading to a flexible component based architecture.
- **Flexible user interface design:** by enabling flexible graphical user interface design on Symbian OS, Symbian is fostering innovation and is able to offer choice to manufacturers, carriers, enterprises and end-users. Using the same core operating system in different designs also eases application porting for third party developers.
- **Robustness:** Symbian OS maintains instant access to user data. It ensures the integrity of data, even in the presence of unreliable communication and shortage of resources such as memory, storage and power.

Symbian OS's major advantage is the fact that it was built for hand-held devices with limited resources that may be running for months or years. There is a strong emphasis on conserving memory, using Symbian-specific programming idioms such as descriptors and a clean up stack. Together with other techniques these keep memory usage low and memory leaks rare. There are similar techniques for conserving disk space (though the disks on Symbian devices are usually flash memory). Furthermore, all Symbian OS programming is event-based and the CPU is switched off when applications are not directly dealing with an event. This is achieved through a programming idiom called active abjects. Correct use of these techniques helps ensure longer battery life.

Unfortunately, Symbian C++ programming has a steep learning curve as Symbian requires the use of techniques that are not common on PC platforms such as descriptors and the cleanup stack. This can make even relatively simple programs harder to implement than in other environments. It is possible that the techniques, developed for the much more restricted mobile hardware of the 1990s, do cause unnecessary complexity in source code; programmers are required to concentrate on bug-prone low-level routines instead of truly applications-specific features. It is difficult however, to make a move towards a more high-level and modern programming paradigm in Symbian, because the platform is so tightly bound to semi-obsolete thinking models about mobile software development.

3.2 Competition

Symbian OS EKA2 [3] also supports sufficiently-fast real-time response. It is possible to build a single phone in which a single processor core executes both the user applications and the signalling stack. This has allowed Symbian OS EKA2 phones to become smaller, cheaper and more power efficient.

3.3 Architecture

Symbian OS architecture is designed to meet a number of requirements (Figure 3.1). It must be hardware independent so it can be used on a variety of phones types, it must be extendable so it can cope with future developments and it must be open to all to develop for.

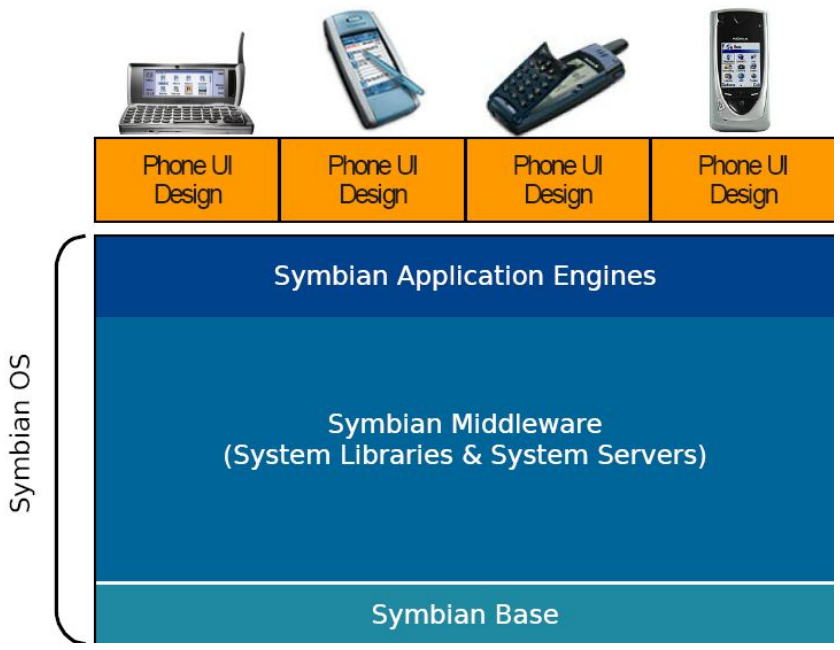


Figure 3.1. Symbian OS architecture

- **Base:** Symbian OS base is common to all devices, i.e. kernel, file server, memory management and device drivers. Above this base, components can be added or removed depending on the product requirements.
- **System layer:** the system layer provides communication and computing services such as TCP/IP, IMAP4, SMS and database management.

[3]EKA2 is the second-generation kernel for Symbian OS.

- **Applications engine:** above the System layer sit the Application engines, enabling software developers (be they either employed by the phone manufacturer or independent).
- **User interface software:** it can be made or licenced by manufacturers (for example in the case of the Nokia Series 60 platform).
- **Applications:** they are slotted in above the User interface.

Chapter 4

Software paltforms

4.1 Nokia implementation

Nokia develops and maintains several advanced software platforms that enable different user interfaces and displays, product concepts, and feature configurations (see Figure 4.1).

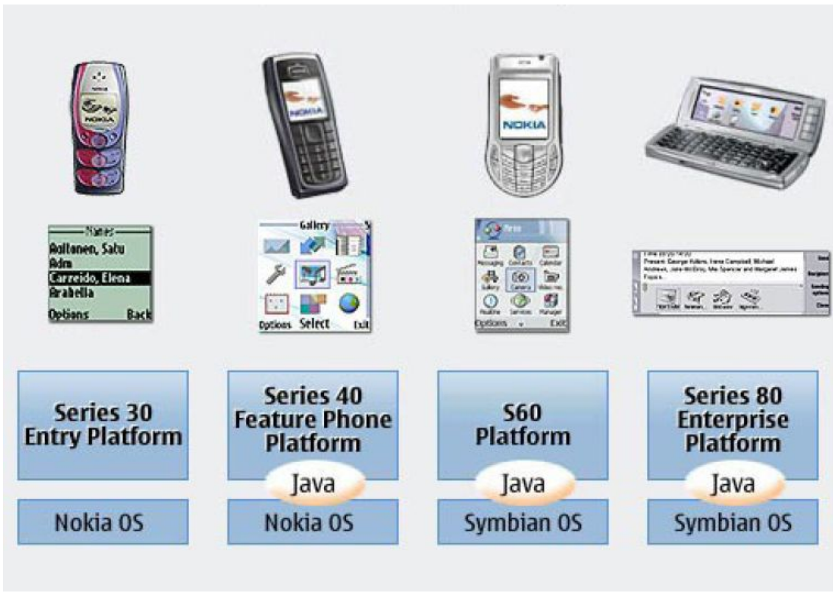


Figure 4.1. Nokia software platforms

- **Series 30:** is the lowest-cost platform, designed for entry level mobile phones that feature voice and basic messaging functionalities.

- **Series 40:** is a versatile, efficient and highly cost effective feature phone platform, particularly suited for the mobile phone product range.
- **Series 60:** is the world's leading rich smartphone software platform [7], available for OEM [1] licensing.
- **Series 80:** is a high-end software platform optimized for enterprise communicators and smartphones, enabling a two-hand operated QWERTY [2] keyboard and a wide screen.

4.2 Nokia Series 60 3^{rd} edition application development

The S60 platform is the market-leading smartphone platform built on Symbian OS. It incorporates all key mobile technologies expected by increasingly sophisticated enterprise users and consumers and it provides revenue opportunities for the full range of stakeholders in the mobile marketplace. As the S60 platform has developed it has raised the bar on smartphone feature provision by taking the lead in developing and implementing many innovations [8].

S60 platform has different editions. It has experienced 1^{st} Edition, 2^{nd} Edition, and the latest 3^{rd} Edition. In each of the edition, it also introduces different feature packs, which incorporate some of the advanced features on each release. Symbian OS is based on open standard, which makes the development on Symbian/S60 open for the developers. The development for the S60 platform is backward compatible although there was a break between the 2^{nd} Edition and the 3^{rd} Edition, which was introduced by the platform security and new compiler used in the platform.

4.2.1 Architecture

Figure 4.2. illustrates the high level architecture of the S60 platform. The platform is based on Symbian OS but also provides additional features [9].

The **Symbian OS Extensions** are a set of capabilities that allow the S60 platform to interact with device hardware functions such as vibration alert, device lights, and battery charge status.

[1]Original Equipment Manufacturer an organization that sells products that are made by other companies.

[2]QWERTY is the most common modern-day keyboard layout on English-language computer and typewriter keyboards.

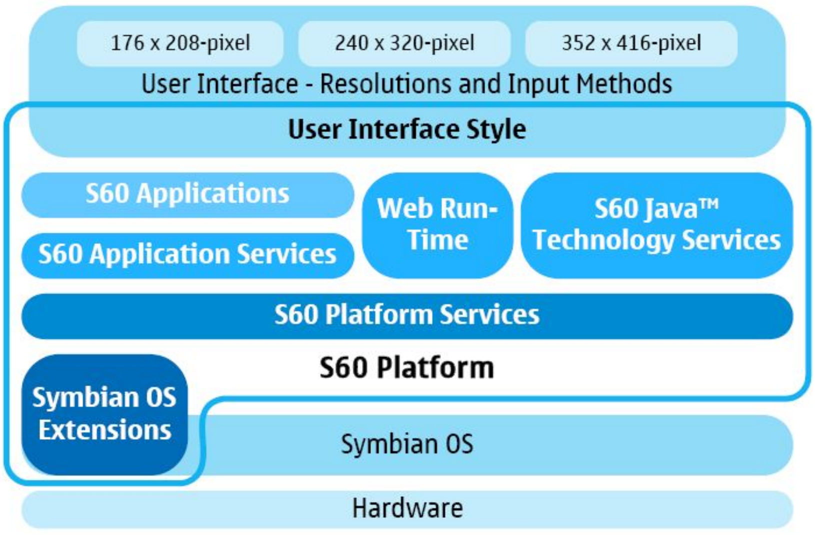


Figure 4.2. S60 platform architecture [10]

S60 Platform Services are the fundamental services provided by the S60 platform, these include:

- *Application Framework Services:* providing the basic capabilities for lounching applications and servers, state persistence management, and UI (User Interface) components.
- *UI Framework Services:* providing the concrete look and feel for UI components and handling UI events.
- *Graphics Services:* providing capabilities for the creation of graphics and their drawing to the screen.
- *Location Services:* allowing the S60 platform to be aware of a device's location.
- *Web-Based Services:* providing services to establish connections and interact with Web-based functionality, including browsing, file download, and messaging.
- *Multimedia Services:* provodong the capabilities to play audio and video, as well as support for streaming and speech recognition.
- *Communication Services:* providing support for local and wide are communications, ranging for Bluetooth technology to voice calls.

S60 Applications Services are a set of capabilities that are used by the S60 applications and can be used by third-party developers to provide basic functionality for applications.

S60 Java Technology services support the $Java^{TM}$ Platform, Micro Edition ($Java^{TM}$ ME) Java Technology for the Wireless Industry (JTWI) specification.

S60 Applications are available to a device's user, include personal information manager (PIM), messaging, media applications, profiles, etc.

The S60 platforms defines **UI style and API's**, but it does not mandate the screen size or input methods. Licensees are free to implement their own customized UIs. Developers must program UI applications with scalability in mind because specific UI dimensions cannot be assumed.

4.2.2 Development process

The development process itself is very similar to desktop C++ or Java development. The process is presented in Figure 4.3.

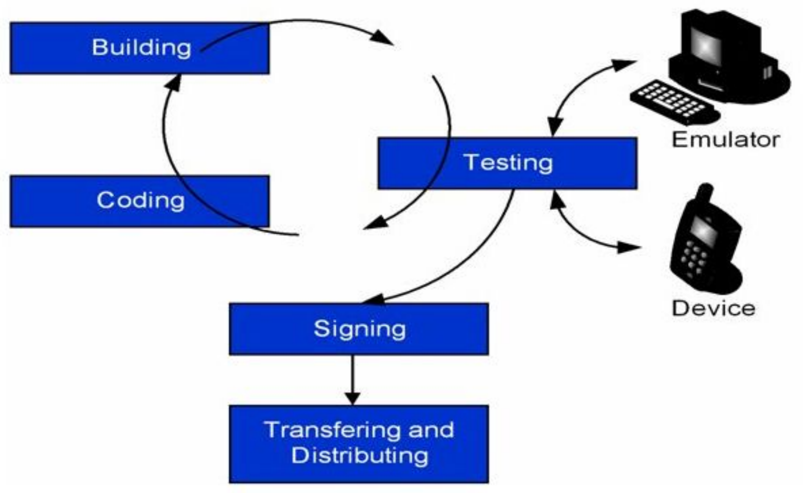


Figure 4.3. Development process for S60 3^{rd} Edition [10]

- **Building in C++:**

  In C++ development, builds are tested in two different environments: in the emulator and on the device. Typically, builds are first tested in the emulator and then final testing is performed on the target device. Because the operating systems in these environments are different,

there are two different types of build targets. The build for the device use another type of compiler. The actual type of compiler used when creating builds for devices depends on the device and project requirements.
Once the build and test cycles have been completed, then the application can be signed.
The platform security model requires mandatory *.sis* file signing. The build process can be done in a command line or in an IDE (Integrated Development Environment).

- **Testing in the emulator:**
  C++ developers should be aware of the fact that altough the emulator is an extensive tool for application testing, it's functionality differs from the devices. This is quite natural because the devices vary and some features are device specific, like memory capacity.

  The functionality differences include:

  i. Because the emulator uses threads and not processes like the device would use, bad pointers can exist.
  ii. The emulator uses default or set heap and stack sizes, which are device specific in the actual devices.
  iii. The emulator has the whole memory capacity of the test PC, but the device has the limited amount of embedded memory.
  iv. The processor capacity of a normal PC exceeds the processor capacity of a target device. This can make big differences on actual performance of an application.
  v. The emulator can be run without platform security, which is not possible in mobile devices based on Symbian OS v9.1 and later.

  Java developers can also test their applications with the emulator included in the Java SDK (Software Development Kit).

- **Signing process:**

  It is compulsory to sign all S60 applications before deploying the application onto a real devices. There are two kinds of signing:

  i. Self-Signed: for applications using UID's between 0x80000000 - 0xFFFFFFFF, which belong to the unprotected range and the application, can only use some of the basic set of capabilities.

ii. Symbian Signed: for applications using UID's between 0x00000000 0x7FFFFFFF, which belong to the protected range and the application, may use some of the sensitive API's.

- **Distributing:**

  Several transfer methods can be used: WAP, e-mail attachment, infrared, Bluetooth, USB. Over-the-Air (OTA) is also suppoerted for commercial distribution.

4.3 The structure of a Symbian OS application

The S60 App Wizard (supplied with the S60 SDK) is the recommended way to start a new application project. The wizard creates an empty application with a menu, which is ready to add features and run. Figure 4.4 demonstrates the structure of a Symbian OS application [12].

The structure contains the following classes:

- The **application class** is responsible for seting up and executing the application. It supplies a globally unique 32-bit identifier (UID) which is always associated with the application (both in the build project and at run time). Changing the UID during the lifetime of the project is not advisable since it is used in a number of places through the application source files.
- The **document class** is created by the application class. It usually has strong ownership of application data and is responsible for persisting and iternalizing the data. This class also instantiates the application user interface (AppUi) class. It is feasible for the document class not to implement anything other than creating the application's instance of the AppUi.
- The **application user interface** is used for event handling. The AppUi acts as a global event, and command, handler. It processes key presses and menu selections and can pass these events on to the views and container classes the make up an application. The AppUi is a controller that has no visible presence on the screen.
- **Views and container classes** provide the screens of the application. They are handled by the view architecture. A view is essentially a container class associated with an ID. A particular view can be activated

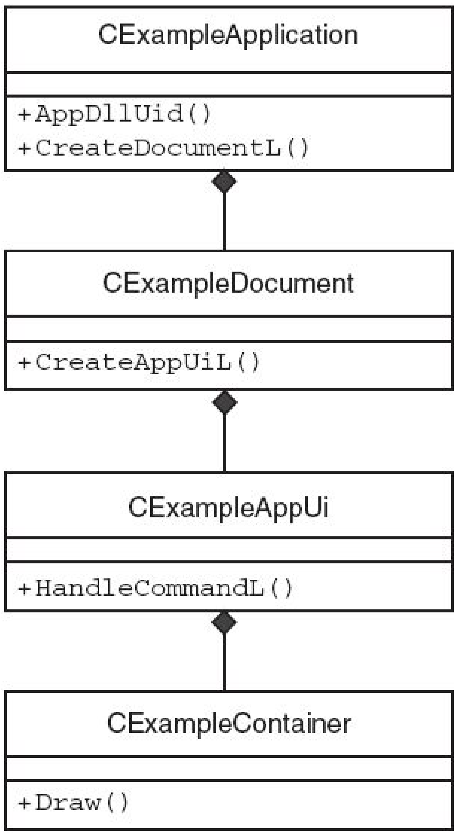


Figure 4.4. Application structure

from within the application or from another application by supplying the UID of the application and the ID of the view.

Figure 4.4 shows the minimum number of classes that need to be created to run an application. More classes may be added as the application evolves and other screens or views are needed.

4.3.1 Directory structure

A standard directory structure for an application is recommended, in order to maintain clarity as the application evolves. If we use a common strucure for the project we will have at least six folders and and a build descripton files:

- The **application interface** (`/aif`) directory contains two icons and their marks. One icon is, a 42 x 29-pixel image, is for the main menu and a larger 44 x 44-pixel icon is for the application's context pane. A resource file is also stored here for the icons.

- The **data** directory hold files that define the specific resource utilized by the application. For applications using Symbian OS v9 and higher, these files address the new security implications.
- The **group** directory holds the resource files for each of the compilers, a UID source file, the `bld.inf` file (which tells the compiler wherethe project file is) and the project file itself.
- The **include** directory holds the header files, global files and localization files.
- The **install** directory contains the package files (`.pkg`) and Symbian installation file (`.sis`). The package file lists all resources and files required by the project. When compiled, using the `makesis` command, the installation file is created.
- The **source** directory stores all the source files (`.cpp`) which implement the functionality of the system. Other folders to storage graphics and sounds may also be added here.

4.3.2 Project file structure

The `.mmp` file is the project definition file. This file lists and describes each of the individual source code files and libraries required by the application or component, as well as directory locations and other project-defined assets. This file can be used to generate the appropriate project files for your chosen IDE, and so allows for a software-independent definition of the application.

The project file defines all the components that the project requires, source files, bitmap and library files are examples of a view. As an example, our applications *PodNet.mmp* file is given in the Appendix B.

4.4 Used technologies

These subsections contain the programming technologies used to port the C++ code to the Symbian platform.

4.4.1 Symbian C++

Symbian OS was built using C++ and this development language provides the fullest access to the feature-rich S60 platform. All versions of the S60 platform suppor C++ development and various integrated development environments (IDEs) are available for each version.

Symbian C++ development for the S60 platform requires two components: a suitable development environment and an sofware development kit (SDK) for the edition and feature pack(s) at which the application will be targeted. SDks are provided free of charge.

Symbian C++ development for the S60 platform is supported by Carbide.c++, a member of the Carbide family of tools from Nokia. Carbide.c++ is based on Eclipse IDE and is being made available in four versions: Carbide.c++ Developer, Carbide.c++ Professional, Carbide.c++ Express and OEM editions.

Carbide.c++ Developer was used in podcast application development, that is designed for application developers with more stringent application-quality requirements. It provides the ability to perform on-device debugging and includes a UI design tool that helps simplify S60 development.

4.4.2 P.I.P.S.

P.I.P.S is termed to be **P**.I.P.S **I**s **P**OSIX on **S**ymbian and it is the first stage in the implementation of POSIX (*Portable Operating System Interface for uniX*) standard libraries for Symbian OS. Also P.I.P.S is termed to be a subset of POSIX libraries. The implementation of these libraries onto Symbian OS will make it easier for C developers in order to develop and build or port their applications to Symbian and as a platform independent one.

POSIX basically contains API's that are compatible across many Operating Systems.

P.I.P.S includes four standard libraries and they are listed below:

1. **libc:** Includes standard C libraries - I/O routines, database routines, Bit operator, String Operator, Character Test and Character Operators, Storage Allocations and Signal Handling.

2. **libpthread:** Thread creation and destruction, interface to thread scheduler, mutex.

3. **libm:** Arithmetic and Mathematic functions as per Standard C library.

4. **libdl:** Loading of Dlls.

The main entry point of P.I.P.S application is similar to normal C application `main()` and not as native Symbian C++ entry point - `E32Main()`.

Advantages of P.I.P.S:

- Can port existing middleware to Symbian OS.
- Can port existing C applications to Symbian OS.
- Developers can create their applications on Symbian OS by using C libraries natively.
- There are lots of Linux Open source projects written on POSIX.

Disadvantages of P.I.P.S:

- User cannot create UI's using P.I.P.S for their applications.
- Symbian P.I.P.S stdio is different from old stdio and user cannot make use of the old stdio library.

4.4.3 Open C

Open C is a set of Industry Standard C libraries and it is an extension of P.I.P.S. Open C brings to S60 well-known C libraries, including subsets of POSIX, OpenSSL, zlib, and GLib.

Open C extends PIPS to include libz, libcrypt, libcrypto, libglib, and libssl. These will effectively required porting a large number of existing Linux applications (see Figure 4.5).

Apart from the above four libraries explained in P.I.P.S., Open C also includes the below given libraries:

1. **libz:** Zlib compression library which provides memory compression and decompression functions which in turn includes integrity checks.
2. **libcrypt:** Cryptography libraries - encrypting blocks of data and messages.
3. **libcrypto:** Used by Open SSL implementations of SSL, TLS and S/MIME and also been used to implement SSH, OpenPGP and other cryptographic standards.
4. **libglib:** General purpose utility library that provide macros, data types, type conversions, string utilities and file utilities.
5. **libssl:** Transport layer security protocols.

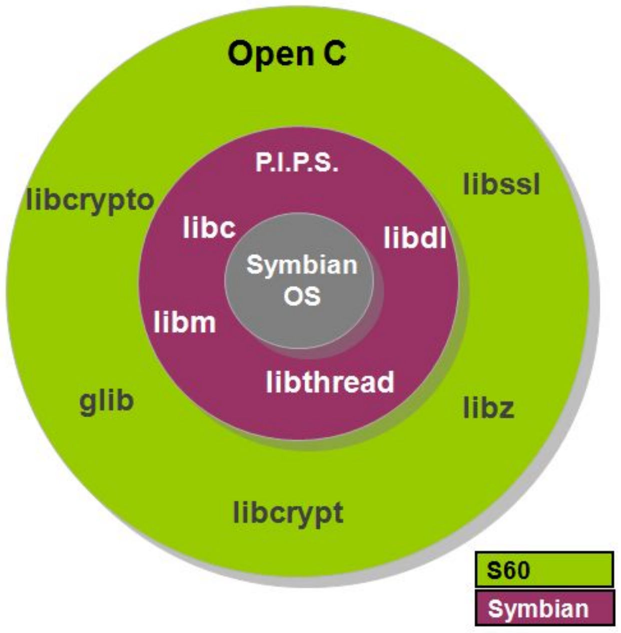


Figure 4.5. P.I.P.S. and Open C important libraries

4.4.4 STLport

STLport is a high-quality implementation of the C++ Standard Template Library (STL), a library that is part of the ISO/IEC C++ Standard and that is not included natively in the Symbian OS platform. By supporting the STL, Symbian applications can benefit from a wide range of existing C++ code that can now be ported easily.

The STLport works on all Symbian OS 9.x devices, both S60 and UIQ variants. The library has been succesfully compiled with (and binaries are provided for) following platform SDKs:

- S60 3^{rd} Edition 1.1 Maintenance Release
- S60 3^{rd} Edition Feature Pack 1
- UIQ 3.0

The only dependency is a conforming Standard C library, that is provided by either the P.I.P.S. library or Nokia's Open C plugin.

Chapter 5

Implementation

This chapter explains all important details of the current implementation. Clements Wacha has written in an earlier master thesis the core program in C++ [4]. Although it was thoroughly tested on Windows CE, it also compiles and works on Windows XP/2000, Linux and Mac OS X. The PodNet core application is based on the measurement application developed in an earlier thesis [13]. The PodNet application was ported to the Symbian platform and the existing software was extended with a GUI that was implemented in Symbian C++.

5.1 Platform independence

The podcast application makes a lot of use of low-level system calls which introduce problems when porting the application to another platform. The original measurement application used a simple wrapper to overcome this problem.

The three difference platforms of PodNet application is presented in Figure 5.1. The core program and the part of Windows Mobile platform has been implemented by Clements Wach. In this thesis we have extended the core application with the Symbian platform independence libraries and a Symbian GUI.

The following system calls are platform specific where we need include the following libraries and capabilities: (see Appendix B)

- threads in general: `libpthread.lib` library
- semaphores and mutexes to protect critical sections: `libpthread.lib` library

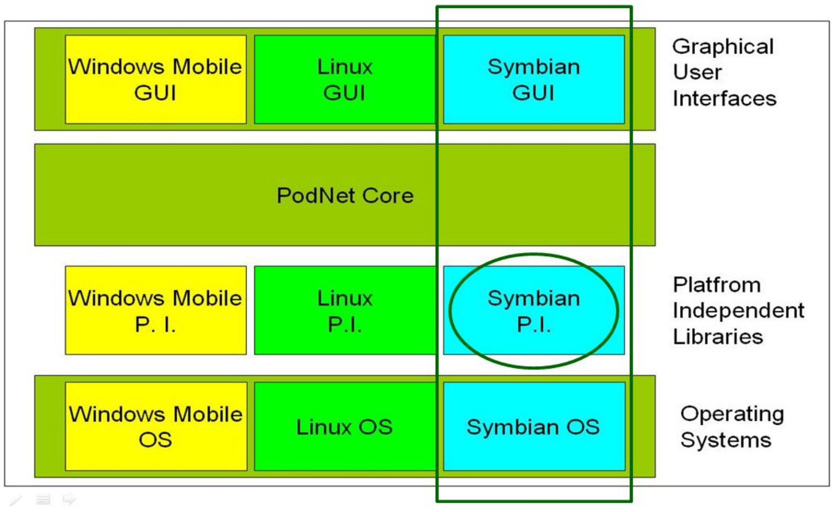


Figure 5.1. The platforms of PodNet program

- network access: `esock.lib` library and `NetworkServices` capability
- file system access other than open, read, write and close: `ReadDeviceData` and `WriteDeviceData` capabilities.

All these functions are collected in the PI (Platform Independence) library which Clements Wacha has originally created in [13] and extended in this work to Symbian by adapting the platform independence library.

5.2 Graphical User Interface

Model View Controller

To make the Graphical User Interface (GUI) optional we had to separate it from the main application core in a clean way. This was done by using the Model-View-Controller (MVC) approach [1].

The view architecture allows to register views and switch between them, with one view running at a time. This requires the use of unique identifiers (UIDs) with each view requiring its own UID.

Three Views have been created:

[1]See `http://wiki.forum.nokia.com/index.php/Design_Patterns_in_Symbian` for a good explanation.

- `CPodNetGUIListBoxAllChannels`: in this view we can see all available channels.
- `CPodNetGUIListBoxDetails`: show all available episodes within a channel.
- `CPodNetGUIListBoxEpisodes`: this screen shows the episode meta information.

A common approach allows each view class to be initiated in the application user interface and this user interface class is the main event handler. Control can be passed down to specific view classes that own a container, which is where the functionality of the view is implemented (see Figure 5.2).

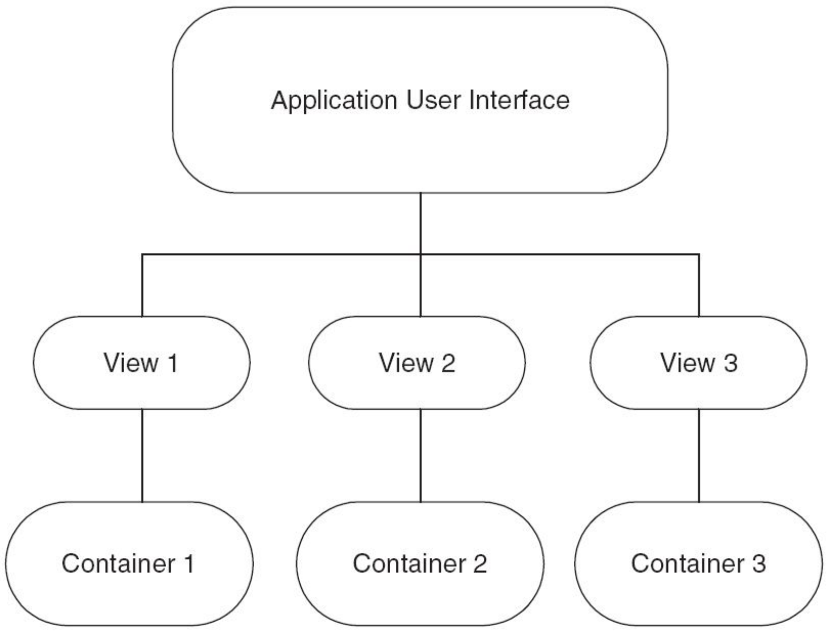


Figure 5.2. Multiple views architecture

In every view, we created a *ListBox*. It was the easiest way that we can insert new items in a list. For example in the first case when we show the *All available channels* in a list box, the list box item is the title of channel.

You can switch between the views if you press on the selected list box item with the middle button or choose the correct menu item (*All channels, Show episodes, Show info*) from the menu list.

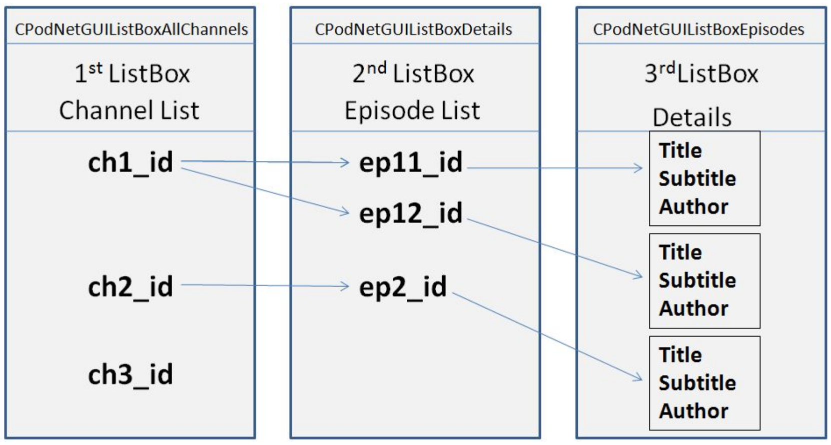


Figure 5.3. GUI architecture

5.3 Conversions

In many cases, we need convert our Symbian descriptors to Open C strings and vice versa because the GUI that had been developed in Symbian C++, communicates with the core application that had been written in C++.

Converting Open C string to HBufC16 descriptor

This way is relying on a dynamic buffer, using *HBufC16* for instance:

```
TPtrC8 pt(reinterpret_cast<const TUint8*>(get_episode.c_str()));
HBufC* temp_get = HBufC::NewLC(pt.Length());
temp_get->Des().Copy(pt);
```

Converting TDesC8 descriptor to Open C string

This way had been chosen, using *char** for instance:

```
TInt length = aCustomMessage.Length();

HBufC8* buffer = HBufC8::NewLC(length);
buffer->Des().Copy(aCustomMessage);

char* episodes = new(ELeave) char[length + 1];
Mem::Copy(episodes, buffer->Ptr(), length);
episodes[length] = '\0';
```

Converting HBufC16 to HBufC8

This example is conversion from 16-bit to 8-bit data where we are just copying:

```
HBufC8 *get = HBufC8::NewLC(temp_get->Length());
get->Des().Copy(*temp_get);
```

Chapter 6

Evaluation

6.1 Testbed

The ad hoc network consisted of Nokia N95 and Nokia N93 mobile devices. Both devices communicated over their integrated WLAN interface. The podcasting application core is written in C++. The graphical user interface is written in Symbian C++.

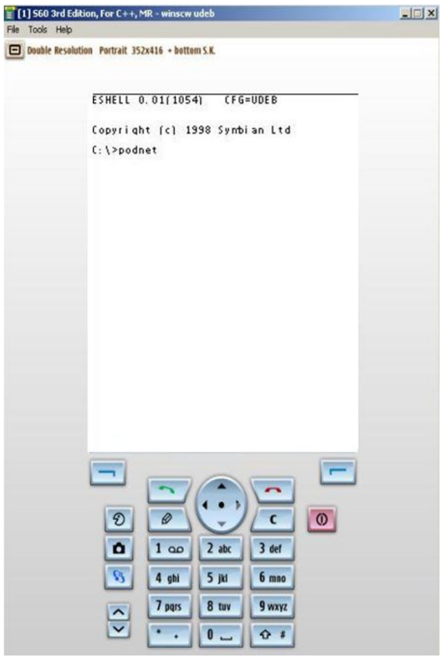
6.2 Testing process

Before we tested on the real devices we used the Symbian OS emulator that is a Windows application called *epoc.exe*, which simulates phone hardware on the PC. The emulator enables software development for Symbian OS to be substantially PC-based in its early stages, although the final development stages will require the use of phone hardware. After the command line testing, we created a Symbian C++ GUI.

6.2.1 Testing on emulator

As first step, the PodNet core application had been compiled and tested like a command line project. In Carbide.c++, the emulator path had been changed to point at a file called *eshell.exe*. The core application can be tested in this command line (see Figure 6.1).

A lot of commands had been tested in command line mode in the emulator but we could not connect to the access point because the emulator does not provide a WLAN interface. Because of that, we were not able to test the commands that need the network access capabilities like `my_ip`.


[1] S60 3rd Edition, For C++, MR - winscw udeb
File Tools Help
Double Resolution Portrait 352x416 + bottom S.K.
1
2 abc
3 def
4 ghi
5 jkl
6 mno
7 pqrs
8 tuv
9 wxyz
0
C

```
ESHELL 0.01(1054)    CFG=UDEB

Copyright (c) 1998 Symbian Ltd
C:\>podnet
```

Figure 6.1. Carbide.c++ emulator in command line mode

6.2.2 Testing with GUI

Channels and episodes with meta information had been created for testing. The list boxes had been tested with these test items:

```
//for example one channel
DS_Channel* chan1 = new DS_Channel();
chan1->updated = PI_GetTime();
chan1->channel_id = "ch1_id";
chan1->title = "channel1";
app->datastore->Channel_Add(chan1);

//for example one test episode (with meta information)
DS_Episode* ep1 = new DS_Episode();
ep1->updated = PI_GetTime();
ep1->episode_id = "ep1_id";
ep1->title = "episode1 ch1";
ep1->subtitle = "ep1 subtitle";
ep1->author = "ep1 Author";
app->datastore->Episode_Add(chan1->ref_id,ep1);
```

We used with test data for the following tasks:

- list available channels (with *Display/All channels* menu item)
- list channel episodes (with *Channel/Show episodes* menu item)
- show the episode meta information (with *Show info* menu item)

Test episodes had not been created under the *channel4* because of if you click on this channel item you can see the *No epsiodes found* listbox item in the episode list box (see Figure 6.2).

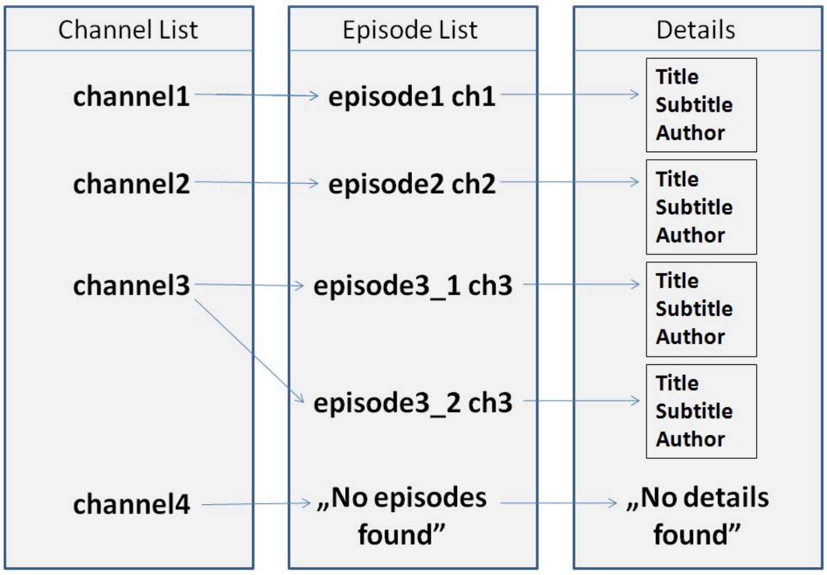


Figure 6.2. Datas for testing

In the 3rd view (*Details*), the right soft key is `BACK`. The handling is not working correctly yet. In the 2nd view (*Episode List*), this menu item is not fully functionality.

The following menu items are working:

- in 1st screen: *All channels, Show episodes and Exit*
- in 2nd screen: *Show info and Back*
- in 3rd scrren: *Back*, is not working correctly.

Chapter 7

Conclusions

This master thesis consists of three part:

- port the existing podcasting code to Symbian by adapting the platform independence library
- extend the existing software with a GUI
- testing of this application on Nokia N93 and N95 handhelds.

The existing podcasting code has been designed and implemented in C++ from an earlier project. This software uses peer-to-peer synchronization to exchange content directly between neighboring devices. We have ported this existing podcasting code to Symbian by adapting the platform independence library. We have tested this program in Carbide.c++ using the command line emulator. Then we have developed a GUI in Symbian C++ that communicates with the core application. Our programs runs on Nokia N93 and N95 handhelds. We have tested and used the application under Windows XP.

We have created channels and episodes for testing. We can show all available channels, episodes within a channel and details of these episodes when the corresponding menu items has been choosen.

Our future plan is to develop the GUI that allow users to subscribe to channel, to publish new content to channels, and to listen or watch downloaded channel contents. Furthermore, extend the existing software with a tracing module that traces the WLAN link conditions between PodNet devices and the GPS coordinates of the devices.

Appendix A

Task Description

This appendix contains the official task description of the thesis project is presented.

The goal of this thesis is to develop a prototype content distribution application for Symbian phones which makes use of their ad hoc capabilities (Bluetooth, Wifi) in order to directly connect them. The application should allow users to exchange pictures and videos that were generated on the mobile phones. Further, as a background process, the application should constantly log all devices that it sees within communication range together with a timestamp and the duration of the cantact. Whenever there is an opportunity, these logs are uploaded to a server at ETH Zurich.

The application should be developed for the Symbian OS [2] since *(i)* Nokia has the highest market share and *(ii)* a partnership with Nokia is under discussion. A major goal of the thesis is to make it run on many devices as possible in order to potentially attract as many users as possible. In order to meet these goals, the thesis is split into the following tasks:

1. *Literature exploration*: Familiarize with the Podnet project [3] and the Master thesis work of Clemens Wacha on wireless podcasting [4].

2. *Identify steps*: Identify the necessary steps to port the existing podcasting code to the Nokia N95 Symbian platform.

3. *Port the code*: Port the code to Symbian by adapting the platform independence library.

4. *Extend the existing software with a GUI*: The GUI should at least allow the users to *(i)* create channels, *(ii)* to subscribe to channels, *(iii)* to publish new content to channels, and *(iv)* to listen or watch downloaded channel contents.

5. *Extend the existing software with a tracing module*: The tracing module should trace the WLAN link conditions (packet loss rates, RSSI values, etc) between PodNet devices and the GPS coordinate of the devices. Extend the tracing module with a mechanism to automatically upload traces to a data sink (a gateway in the Podnet system architecture).

6. *Test the application*: It should test on Nokia N95 and Nokia N93. In addition test the application for compatibility with the Windows Mobile and the N810 Linux implementation.

7. *Setup a Web page*: Create the Web page so that people can install the developed software by means of docking as well as by direct installation without docking.

8. *Thesis writing*: A detailed report of the performed work will be written.

A.1 Working Plan

Table A.1 shows the working plan of this thesis. The task numbers refer to the points defined above.

Week	**Date**			**Tasks**							
				1	2	3	4	5	6	7	8
1	11th Febr.	-	17th Febr.	X							
2	18th Febr.	-	24th Febr.	X							
3	25th Febr.	-	2nd March	X	X						
4	3rd March	-	9th March		X	X					
5	10th March	-	16th March			X					
6	17th March	-	23rd March			X					
7	24th March	-	30th March				X				
8	31th March	-	6th April				X	X			X
9	7th April	-	13th April				X	X			X
10	14th April	-	20th April				X	X			X
11	21th April	-	27th April				X	X	X		X
12	28th April	-	4th May					X	X		X
13	5th May	-	11th May						X	X	X
14	12th May	-	18th May					X		X	

Table A.1. Working plan of the thesis project

Start:	Monday, 11th February 2008
End:	Friday, 9th May 2008

Zurich, 22nd February 2008
Ákos Bakos

Appendix B

Project definition file of PodNet

The *PodNet.mmp* project definition file specifies the properties of a project in a platform and compiler independent way.

```
TARGET PodNetGUI.exe
UID     0x100039CE 0xEAE869F1
VENDORID    0
TARGETTYPE exe
EPOCSTACKSIZE 0x5000

SYSTEMINCLUDE  \epoc32\include
SYSTEMINCLUDE  \epoc32\include\variant
SYSTEMINCLUDE  \epoc32\include\ecom

SYSTEMINCLUDE \epoc32\include\stdapis
SYSTEMINCLUDE \epoc32\include\stlport
SYSTEMINCLUDE \epoc32\include\libc

USERINCLUDE ..
USERINCLUDE     ..\inc
USERINCLUDE     ..\data
USERINCLUDE ..\PI
USERINCLUDE ..\tinyxml

SOURCEPATH ..\data
START RESOURCE PodNetGUI.rss
HEADER
TARGETPATH resource\apps
END //RESOURCE

START RESOURCE PodNetGUI_reg.rss
TARGETPATH     \private\10003a3f\apps
END //RESOURCE

LIBRARY euser.lib apparc.lib cone.lib eikcore.lib avkon.lib
LIBRARY commonengine.lib efsrv.lib estor.lib eikcoctl.lib eikdlg.lib
LIBRARY eikctl.lib bafl.lib fbscli.lib aknnotify.lib aknicon.lib
LIBRARY etext.lib gdi.lib egul.lib insock.lib
LIBRARY ecom.lib InetProtUtil.lib http.lib esock.lib libc.lib libm.lib libpthread.lib
```

```
LANG  01

START BITMAP  PodNetGUI.mbm
HEADER
TARGETPATH  \resource\apps
SOURCEPATH  ..\gfx
SOURCE c24  podnet_icon_w.bmp
END

SOURCEPATH ..\src

#ifdef ENABLE_ABIV2_MODE
DEBUGGABLE
#endif

SOURCE PodNetGUIApplication.cpp PodNetGUIAppUi.cpp PodNetGUIDocument.cpp

SOURCEPATH ..\PI
SOURCE pi_filesystem.cpp pi_socket.cpp pi_string.cpp pi_tcpsocket.cpp
 pi_threads.cpp pi_time.cpp pi_udpsocket.cpp

SOURCEPATH ..\tinyxml
SOURCE tinystr.cpp tinyxml.cpp tinyxmlerror.cpp tinyxmlparser.cpp xmltest.cpp

SOURCEPATH ..
SOURCE analyser.cpp automator.cpp bloom_filter.cpp datastore.cpp debug.cpp
 flexpacket.cpp message.cpp ndishelper.cpp rc_commands.cpp router.cpp
 simple_rpc.cpp sync.cpp transfer.cpp main.cpp

STATICLIBRARY stlport_s.lib

SOURCEPATH ..\src
SOURCE PodNetGUIListBoxAllChannelsView.cpp PodNetGUIListBoxAllChannels.cpp
 PodNetGUIListBoxEpisodesView.cpp PodNetGUIListBoxEpisodes.cpp PodNetGUIListBoxDetails.cpp
 PodNetGUIListBoxDetailsView.cpp
```

Appendix C

Development Environments

C.1 Setting up the Environment

In order to get started we need a couple of tools installed on our PC, these include an Integrated Development Environment (IDE) for building our applications and a suitable Software Development Kit (SDK). The SDK contains Application Programming Interface (API) documentation, example code, and a number of development tools, including an emulator for testing and debugging our application before actually deploying it on the target phone.

The recommended IDE by Symbian and Nokia are currently the Eclipse based Carbide.c++. The Carbide.c++ IDE is available in three different versions:

- Carbide.c++ Express Free version for non-commercial developers
- Carbide.c++ Developer Additional capabilities for commercial developers
- Carbide.c++ Professional For advanced commercial developers

In the following we will be using the Carbide.c++ Developer IDE. It is recommended that we have a fairly fast development PC. For running the Carbide.c++ Developer edition, the following configuration is recommended: Windows XP (SP2) or Windows 2000 (SP4), 1800MHz processor, 1024 MB RAM, and enough free hard drive space for the IDE and a SDK - typically around 650 MB (additional installs such as Java Runtime Environment and Perl are not included in the 650MB, but commonly required).

The choice of SDK to install depends on the target device. As we may have noticed, the UI and input capabilities of Symbian OS based devices can vary from phone to phone. Some phones provide pen-based input while others are designed for one-hand use with a numeric keypad. This flexibility means that we have to use the SDK matching the UI capabilities of our target device. Currently the majority of Symbian phones use the S60 UI. S60 is available in a number of revisions; these are shown in the list below.

- S60 3rd Edition Symbian OS v9.1
- S60 2nd Edition Feature Pack 3 Symbian OS v8.1
- S60 2nd Edition Feature Pack 2 Symbian OS v8.0a
- S60 2nd Edition Feature Pack 1 Symbian OS v7.0s enhanced
- S60 2nd Edition Symbian OS v7.0s
- S60 1st Edition Symbian OS v6.1

Symbian maintains a list of phones using each version of the Series 60 SDKs. Here we can check the device section for the platform and feature pack version of a specific phone model. As mentioned previously, we will be using the S60 3rd Edition Maintenance Release. As seen on the list from Symbian our application will be targeting the following phones: Nokia N95 and Nokia N93. However, there is a large chance that our application will also run on other phones within the S60 3rd Edition Maintenance Release category.

C.2 Installing the IDE

In order to download the IDE we need a Forum Nokia account. If we are not yet a registered member go to registration to create an account first. We will also need to register the SDK so if we do not have an account, now is a good time to get one. In addition to allowing us to register the IDE and SDK for free, we will also be able to use the development-related community forums at Forum Nokia.

The Carbide development tools can be downloaded from Forum Nokia webpage. Choose the free Carbide.c++ Express version. If we want to implement graphical UI design we need the Carbide.c++ Developer version. After installing the IDE let us quickly move on and install a SDK.

C.3 Installing the SDK

The different Symbian SDKs can be found on Forum Nokia's C++ SDK. Be careful to download an SDK compatible with Carbide.c++. This means that we can currently choose between the following SDKs:

- S60 3rd Edition, FP2
- S60 3rd Edition, FP1
- S60 3rd Edition Maintenance Release
- S60 2nd Edition, FP3
- S60 1st Edition, FP1

In our case we will download the 3nd Maintenance Release.

Note that the installation directory must be on the C: drive and its name must not contain spaces (the best thing to do is just to accept the default installation directory).

C.4 Additional installations required

Perl (www.activestate.com)

We also need to install Perl as several of the build scripts in the SDK rely on this.

- ActivePerl-5.6.1.635

Java (www.java.com)

In order to utilize the phone emulator fully, we also need a working installation of Java Runtime Environment.

- Download the latest version from www.java.com/download

Open C (Forum Nokia - Open C)

The provided Open C SDK plug-in is intended for developers porting software using Open C.

- Download the free Open C SDK from www.forum.nokia.com

STLport (www.stlport.sourceforge.net)

We need still STLport, that a high-quality implementation of the C++ Standard Template Library (STL).

- We can download the prebuilt for S60 3rd Edition Maintenance Release SDK

Nokia PC Suite (www.nokia.com)

A final tool that will come in handy is the PC Suite from Nokia. Installing the PC Suite will enable us to easily transfer our application to the actual phone.

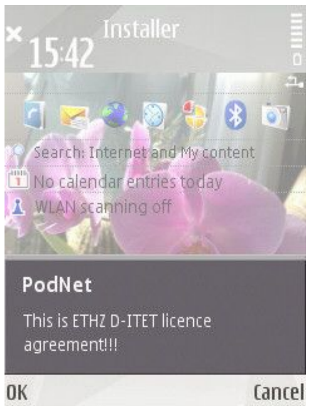
- Download the latest version from http://europe.nokia.com/A4144905

This completes the installation of IDE and SDK - we are now ready to put our tool to use and start creating Symbian C++ applications.

Appendix D

User's guide

Several transfer methods can be used that the PodNet application is sent from development environment to our mobile phone such as via Bluetooth, infrared, USB. After the transfer the application has been installed on the device. Our own licence agreement had been used during the installation (see Figure D.1).



PodNet

This is ETHZ D-ITET licence agreement!!!

Figure D.1. Licence of ETHZ D-ITET

After the installation, the application starts with a startup screen where all available channels are shown in a list (see Figure D.2).

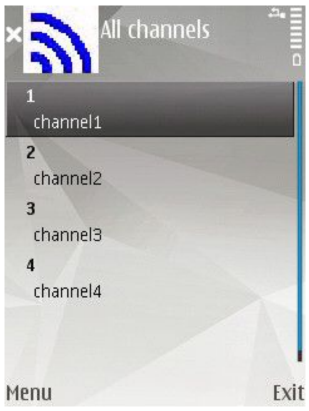


Figure D.2. All available channels

In this screen we can scroll with the arrow button from channel to channel and if we click the *Menu* button, we can choose from the menu items presented Figure D.3.

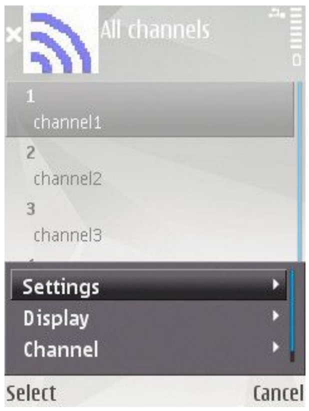
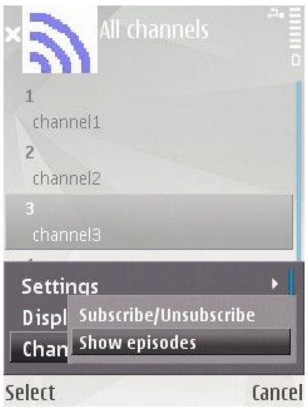


Figure D.3. Menuitems in the startup screen

After this we can press on the selected channel with middle button of mobile device and then we can see the list of channel episodes or we can use the *Show episodes* menu item (see Figure D.4).



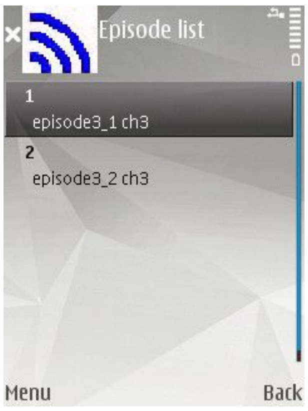
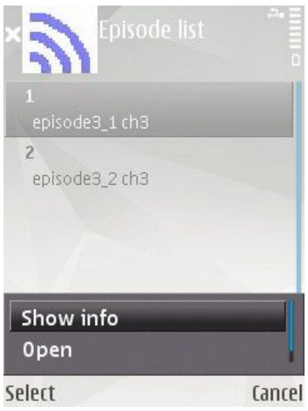
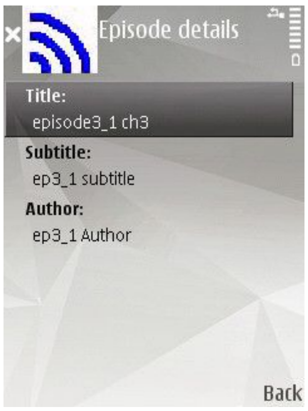


Figure D.4. Show the list of channel episodes

If we click in the right Figure D.5 of the screen on the selected episode, we can see the episode details. In an other case we can see it, if we select one episode and then we press on *Show info* menuitem.





Title:
episode3_1 ch3
Subtitle:
ep3_1 subtitle
Author:
ep3_1 Author

Figure D.5. Meta information of the episode

From this screen you can go back to the episode list screen.

Bibliography

[1] Karlsson G., Lenders V., May M., *Delay-Tolerant Broadcasting*,
In Proceedings of the ACM SIGCOMM Workshops, Pisa, Italy, September 2006

[2] Symbian OS
http://www.symbian.com/

[3] The Podnet Project
http://podnet.ee.ethz.ch/

[4] Clemens Wacha, *Wireless Ad Hoc Podcasting with Handhelds*,
Master Thesis MA-2007-05,
TIK, ETH Zurich, May 2007

[5] Wikipedia, Symbian OS,
http://en.wikipedia.org/wiki/Symbian_os

[6] Jo Stichbury, Mark Jacobs, *The Accredited Symbian Developer primer: Fundamentals of Symbian OS*,
John Wiley & Sonc Inc., 2006

[7] The world's leading smartphone software S60 extends leadership with six new devices
http://www.nokia.com/A4136001?newsid=1190186

[8] S60 3rd Edition: Application Development
http://wiki.forum.nokia.com/index.php/S60_3rd_Edition:_Application_Development

[9] Nokia and Symbian OS, White Paper
http://nds2.ir.nokia.com/NOKIA_COM_1/About_Nokia/Press/White_Papers/pdf_files/symbian_net.pdf

[10] Forum Nokia, Developer Community Wiki
`http://wiki.forum.nokia.com/index.php/S60_3rd_Edition:_Application_Development`

[11] Wikipedia, Podcasting,
`http://en.wikipedia.org/wiki/Podcast`

[12] Paul Coulton and Reuben Edwards with Helen Clemson, *S60 Programming: A Tutorial Guide*,
John Wiley & Sonc Inc., 2007

[13] Trajkovic I., Wacha C., *Ad Hoc Communication with Handhelds*,
Semester Thesis TIK-2006-02,
TIK, ETH Zurich, February 2006