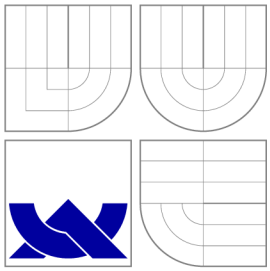
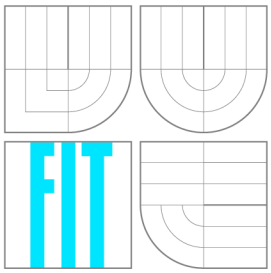


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ASISTENČNÍ A INFORMAČNÍ SYSTÉM PRO ZRAKOVĚ POSTIŽENÉ

ASSISTANCE AND INFORMATION SYSTEM FOR BLIND PEOPLE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — DAVID HNILICA
AUTHOR

VEDOUCÍ PRÁCE — Doc. Dr. Ing. JAN ČERNOCKÝ,
SUPERVISOR

BRNO 2007

Abstrakt

Tato práce řeší problém implementace asistenčního systému pro podporu zrakově postižených osob v prostředcích hromadné dopravy. Jelikož se jedná pouze o část většího celku (projektu RAMPE), není v této práci pokryta celá problematika, nýbrž pouze její část. Práce je psána z pohledu vývojáře a softwarového architekta. Zlepšuje architekturu dříve vyvinutého projektu, navrhuje nové metody a techniky a přidává do výsledné aplikace nové funkce. Toto však platí pouze pro příslušnou část - práce nemůže a ani si neklade za cíl změnit nebo navrhnout celý systém. To už bylo provedeno jinými autory v minulosti a tato práce je na těchto předchozích výsledcích postavena.

Klíčová slova

RAMPE, zrakově postižení, hromadná doprava

Abstract

This work deals with the problem of implementation of the assistance system to support the orientation of blind people in the means of public transportation. Since it is only one part of larger unit (the RAMPE project), it doesn't cover the whole topic, yet only implements one part of it. It is written from the point of view of a developer and code architect. It improves the software architecture of existing project, suggests new methods and techniques and adds new functions to the application. This is done only at the respective part - the work is neither capable, nor takes any ambition to design or modify the whole system. That has been done already by other people and parties in the past and this work merely continues those done in the past.

Keywords

RAMPE project, visualy impaired people support, public transportation

Citace

David Hnilica: Assistance and Information System for Blind People, diplomová práce, Brno, FIT VUT v Brně, 2007

Assistance and Information System for Blind People

Prohlášení

I hereby declare, that I have created this work by myself under supervision of prof. Genevieve Baudoin, Olivier Venard and doc. Dr. Ing. Jan Cernocky. I have also provided all reference, that has been used.

.....
David Hnilica
May 18, 2007

Contents

Chapter 1

Introduction

This work is part of the RAMPE project and has been done under the terms of Socrates/Erasmus programme at the ESIEE Paris under the supervision and with kind help from prof. G. Baudion and Mr. O. Venard from ESIEE Paris and pedagogical lead of doc. J. Černocký from FIT VUT. Whole RAMPE project is being developed by multiple companies and institutions in France. Students work (this work) is supposed to improve and extend the work done before at ESIEE (precise goals are specified further).

1.1 Description of RAMPE project

As described in [5] and [3], the RAMPE project aims to design, realize and experiment a system for the assistance and information of blind people so that they can increase their mobility and autonomy in public transport. It is intended to be deployed in bus or tramway stops or to be installed around the nodes of transport interactions. It is based on smart hand-held Personal Digital Assistant (PDA) with embedded speech synthesis and it is able to communicate primarily via a wireless WiFi connection (other types are considered for future) with fixed equipment in the bus or tram stations.

1.1.1 Introduction into the topic

Orientation in the means of public transportation may prove to be difficult for blind people, as most of the informations given to the passengers relies heavily on the visual information channels.

In the past there had been several approaches taken in order to solve this problem, yet each of them had certain necessary flaws. In France it was mainly the project PREDIT (more in [7]), that had been taken in order to test and evaluate of various systems related to the problem. Amongst other things, it had been found, that one of the main problems is the the lack of interactivity and adaptation of the assisting system to the user and the environment. RAMPE has the ambition to fill such a hole.

1.1.2 Parts of RAMPE

RAMPE project may be separated into three main parts:

A mobile device (client) carried by the user. The device needs to have a WiFi adapter and must be running the the RAMPE application software.

Base stations (a.k.a. "**Bornes**") installed at the bus stops. Those act as servers providing information to the client applications via WiFi link (so far - may change in the future) and are also equiped with a loudspeaker to reveal their position to the blind persons. They also need to have a link to the central system.

A central system synchronizing the vehicles, base stations and the information system of the transportation company. The provided informations may change as the project will grow, but so far it is mainly informations about the available lines, their schedules and certain exceptional events (e.g. delays, repairs, etc.).

1.1.3 Key features of RAMPE

RAMPE introduces an exceptional adaptability by allowing a dynamic change of presented informations instantly. By periodic updates of the available informations it attempts to match the difference in amount of informations (related to the transportation), that is lost by the visual impairment of the user.

While designing the application, special care has been given to the accessibility design of the man-machine interface and to the priorities management of informations provided to the user in the real-time. The main characteristics of the (client) application on the PDA are:

- it can present vocal messages
- it is able to adapt itself to the type of information system available at the particular station (if there are multiple types available)
- it acts upon the actual informations sent by the station

The equipment installed at the stop point has data (in form of a XML file) stored inside. Those data are downloaded to the PDA and provide following informations to the user :

- destinations of the lines at this stop
- waiting time for the bus arrival
- names of bus-stations on a line
- possible disturbances (repairs, temporary changes in schedule, etc.)

The typical use of RAMPE can be demonstrated on picture 1.1.

1.2 Position of this work within RAMPE project

RAMPE is a rather large project (as described further), where many parties contribute. Role of the work on ESIEE in the whole project is the implementation of the client application - the part, that is actualy being used by the blind person directly. Other parts of RAMPE (like the structure of the whole network, details of the equipment used at the stops or reasons and aims of the system specification) are at this work taken mostly as fixed inputs to this work, and as such should not be affected (at least not directly in this work), therefore any mention of those parts, that is made here, is done only for the purpose of the description of the larger context and to help the reader to understand the topic easier.

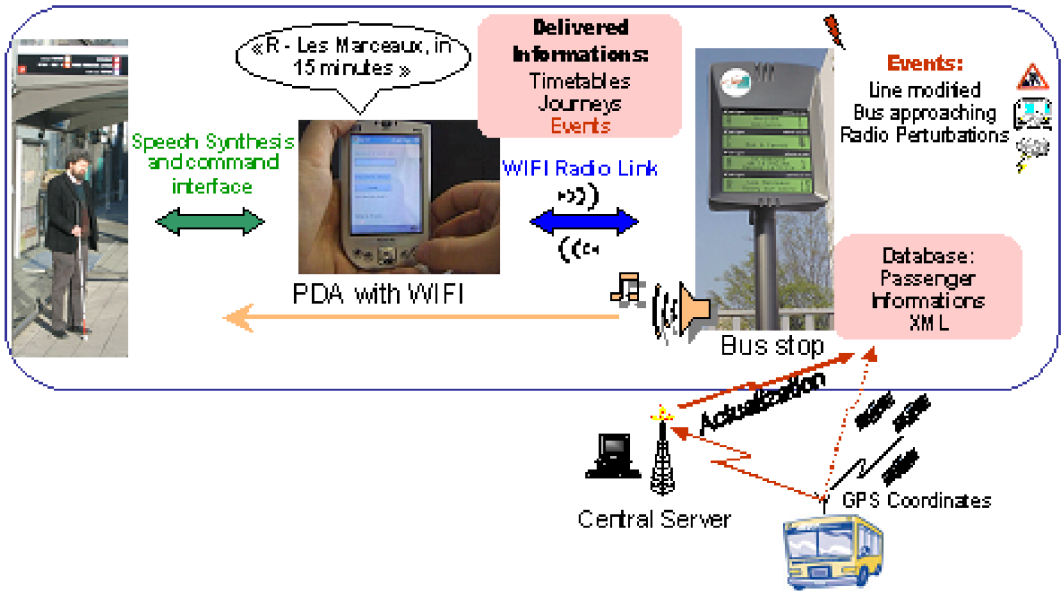


Figure 1.1: RAMPE project

1.3 The role of the author and this work

This particular work was supposed to be improving existing code as follows (quoting informations given before the start of internship):

- improving the software architecture (priority)
- extending application reliability
- adding new features and functionality

On the place after determining the actual state of project, minor changes in the objectives took place. As the point "improving the architecture" would prove to be difficult, while preserving the given code, it was decided, that as a first priority, the application needs to be redesigned (and the code refactored), reaching the other goals either while doing so, or later if possible.

The first thing to do was redesigning the original concept (described further) and implementing the design, so that the resulting application would be functionaly equivalent (or better) to the original one.

1.4 Parts taken from the previous phases of the project

Due to the reasons stated further in the respective chapters, it was difficult to reuse the code from previous phases of the project.

It has been decided instead that the source code of the whole application should be refactored according to the original specifications, written in the [4]. Therefore the only things taken from the original code are ideas - not the code itself.

It has been discovered later during development, that previous author's had sometimes used the same sources of informations, as had the author now (e.g. code examples from

MSDN -[2] and various other websites). Wherever such a code is used in larger scale (namely XML processing and WiFi driver handling), it is stated in the code in the appropriate form - usually a comment at the begining of the header (if whole class had been used), or at the begining of the used part (if inside a function definition). It is not stated further in this document, since it is trying rather to describe general ideas, than the code that implements them.

1.5 Structure of this work

The structure of this work is keeping the standard defined for the Master Thesis at FIT VUT.

Description of the chapters follows.

Specification and requirements gives closer specification (software and hardware) of the platform and development environment is given in this chapter.

Goals and priorities is trying to specify the goals of development and describe the requirements for the application, its code and design. It also attempts to describe the different priorities (from the common "professional" ones), that had to be kept, given the possible varieties future developing community.

Main modules of the project is a brief description of the most vital parts of project. Given the size of this work, this is done only in a very general matter, merely describing what the particular module does (skipping any more precise, yet overly too long mentions about implementation).

Supporting parts is very similar to the previous chapter. The only difference is, that this chapter covers those modules, that yet they do not have much influence on the functionality, they are vital from the developers point of view.

Open issues is trying to summarize the biggest flaws of the project design.

Conclusions is the final chapter of this work. It summarizes the results reached and the possibilities for future.

1.6 Links to any previous author's work

This work is a direct continuation of the previous semester project of the same name. There is no other link between this work and any other previous work of the same author.

Chapter 2

Specification and requirements

2.1 Hardware platform

2.1.1 PDAs

RAMPE project (precisely - the client part developed at ESIEE) is supposed to able to be used on as many kinds of PDAs and smartphones as possible. At this phase however, the only hardware available so far is a some small range of HP IPAQ (Pocket PC platform) products (originaly 41xx line, nowadays 24xx and newer).

Although all the PDAs used in the project so far have different characteristics, they have to have some things in common (necessary for the successful usage in the RAMPE). They all have:

- WiFi connectivity
- Bluetooth connectivity (not used so far, usage is planned)
- Audio output
- OS: MS Windows CE (ver. 4.2 or higher) or Windows Mobile (ver. 5.0 or higher)

2.1.2 Network equipment

For simulation of the running RAMPE network the project uses commonly accessible WiFi routers (namely those of Linksys WRT 54 XX series), connected to a PC that simulates the software running at the stop point. So far it means tuning the HTTP server with the XML datasheet and a simulator for sending/receiving RAMPE-specific data (navigation of the blind person to the stop, sending urgent messages, ...).

2.2 Software platform

2.2.1 Operating system

As the only PDAs available in the project so far are those made by HP, the selection of used software is delimited by this fact. Although HP as a company is known for supporting alternative operating systems (in the past even having its own UNIX-like OS), this applies almost exclusively on their server (and applications) branch. In the hardware they sell to

the end customer, the situation is quite different. Specially in the PDA product portfolio, the only (officially) supported OS is the one provided by Microsoft.

In spite of the existence of an open-source project trying to replace the proprietary OS in the PDA with something based on a free license (namely Linux under GPL), it has not been chosen to be used (at the moment). So far it has too many drawbacks that effectively prevent its deployment in RAMPE - namely that its considered reasonably stable only on older PDA models and the newer PDAs (those used in the RAMPE project) are not yet supported at all.

2.2.2 Programming language and environment

So far the project has been implemented in Microsoft embedded Visual C++ with MFC (Microsoft foundation class) extensions. It was chosen primarily for its good integration with different versions of Windows CE - an operating system, that all the PDAs available at ESIEE support (mostly the only OS they support).

As MFC is not being developed anymore, there will probably arise the need to switch for some other environment in the future. It cannot be solved immediately, since it needs to be approved by other parties of the project

Chapter 3

Goals and priorities

The main goal of this work has been determined by the fact, that the project is being developed at the university mostly by undergraduate students. Those can have different degree of experience and understanding of programming techniques and the project managers may not be able to review each and every piece of work done by the students.

That determined a lot about the work, that had to be done. Given the fact, that the original code had literally lacked any deeper thoughts of design, new design had to be done. This also allowed to set some new goals. They had been set to correspond not just with the immediate needs of project, but also with the (however relatively small) experience of the author and according to available sources on the internet (for example at [9]) and literature ([6] or [1]). The description of goals follows in order of importance.

3.1 Low coupling

A modification in one part (module) has not to affect (if possible) any other parts.

This was a vital rule - and the highest priority one. This (together with some derived sideeffects) allows distributing the work between multiple students without a need for each of them to consult every change with their peers and supervisors.

3.2 Definition of interfaces for important modules

This is tightly derived from previous "low coupling". Since the modules have to coexist while hiding as much of internal informations as possible, the definition of their interfaces needs to be done and kept.

As a side effect, it also allows interchangeability between two modules with same responsability - e.g. network adapter interface, when implemented for two different physical devices (e.g. WiFi and Bluetooth) may be used the very same way by the application without any knowledge of its precise type whatsoever.

3.3 One responsability per module

Rule, that had perhaps became a mantra of every software designer from the very beginning of this profession. To keep the design simple and easily understandable, it was necessary to strictly set the responsability and scope of a module. No module should do neither more, nor less, than its interface says.

3.4 Complexity through simplicity

Again, motivated by the struggle to provide as easily understandable design as possible, it was decided to separate more complex objects into simple parts (while still keeping the previous rules), rather than trying to build complex things at once. Things are done in the manner, so that if there is no need to keep a large quantity of functions in one module, it is rather splitted to multiple modules, that are simple inside - though put together, they provide the desired complex functionality.

3.5 Ease over effectivity

"premature optimization is the root of all evil"

As it might have been noticed already by the reader, none of those rules yields any gain in getting a fast-running code. This is (again) determined by what was said at the beginning of this chapter - the project is in its early stage and it is more of a "proof of concept" and a testbench for implementing new functions, rather than an "ready to use" application for commercial (or simply public) release.

In that case, famous Hoares quote from above can be effectively applied. The author is not saying, that the code should not be optimized at all - just that the optimalization needs to come only when the project is mature enough and its behavior and functions don not change too often.

3.6 Preserving the original man-machine interface

Since the original application had been tested already, it was crucial to start improving the functionality at the point, where the original application ended. This determined most of the work, that had to be done (from the functional point of view). Therefore the old application was used as a specification of the behavior for the new one.

3.7 Documentation

Since the developers are likely to be changing often on the project, the good documentation is a must in the form of code comments (possibly improved by doxygen), as well as an accompanying texts and reports.

Chapter 4

Main modules of the project

Since the old application has been used as a specification, lots of things had been determined in advance. From the functional point of view, the application could be splitted into a small number of (relatively) independent parts, that cooperate together.

- Man-machine interface
- XML Parser
- Text to speech synthetizer (TTS)
- Network control

For easier understanding of parts and their communication with others the block diagram had been made and is presented in 4.1.

The description of parts follows in the respective sections.

4.1 Man-machine interface

As RAMPE is not supposed to be used by other than blind people, usage of a classical ordinary "GUI" would be pointless. RAMPE however has a GUI dialog of its own, but it does not display other than testing informations and its usage is very limited (practicaly only for catching the keypress events and movement on the touchscreen).

Since the means of communication with the user are greatly reduced in this case, the design of man-machine interface had to be very thoroughful and had been done by a special "ergonomy group", that worked only on the design of such interface (results can be found in [4]).

That is also the reason why this topic will be covered only very briefly just to provide an overview of how the system appears to the user (even though the usage of that word is somehow inappropriate, since "appearance" may resemble something connected with vision - sense that is not available for the user at all).

4.1.1 Output methods

As the aim of the project is to design an application for widely available hardware, perhaps the only method for presenting output to the blind user remains the audio output.

At the moment the sound in RAMPE is being used in two ways.

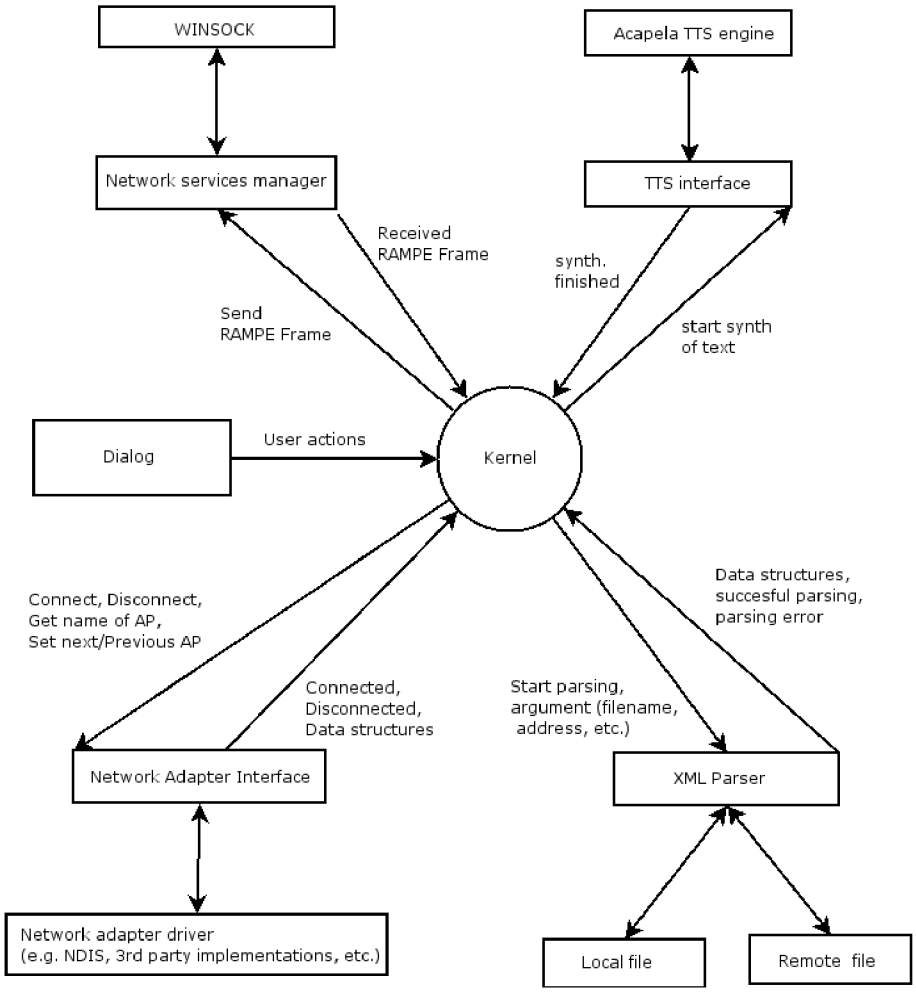


Figure 4.1: Block diagram of RAMPE

- playback of short sounds (beeps, rings, etc.)
- text to speech synthetizer (TTS)

Short sounds are used for presenting to the user common events, that happen often and do not require further specification (e.g. button press acknowledgement, discovering of a new host for connection, error, etc.).

Text-to-speech synthesis (further referred as *TTS*) is then used for providing deeper and more precise informations - e.g. names of lines, names of stops, urgent messages, etc.

4.1.2 Input methods

Since the application runs on a common PDA, it has no other methods for taking users input than a few function keys, touchscreen (for gestures) and audio input.

Out of these, audio input is not used at all at the moment. It is planned for future, but for now no suitable speech recognizing software had been found.

Touchscreen (however it may not seem so) has a great potential, since it can provide the possibility of "gesturing" the required action. It is not easy though, since blind people

are very special group of users and the design of such gestures can not be done in a generic way. Therefore the only usage of gestures at the moment is turning up/down the volume by straight movement up/down. At this part, there is a good potential to be revealed, yet it still requires further research by other participants (the "design group") in the project.

The most widely used input method so far are the hotkeys. Since there is only a very limited number of them and the ergonomy has to be taken into sight (which limits the number of possibilities even more), the design of their usage had to be very careful. Deeper explanation can be found in [4], but for a brief description it is enough to say, that their behavior depends on the current state of the application (based on the motto: *"Apple made it possible to control the whole OS using a mouse with only one button. So we still have about 3 spare buttons..."*). Picture 4.2 shows currently available (and used) scheme of keyboard.

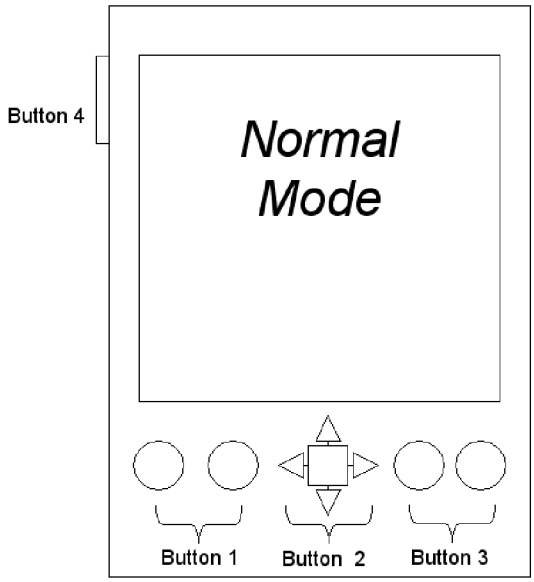


Figure 4.2: usage of PDA hotkeys

4.2 Behavior controller of the Man-Machine Interface

The most popular model of today for a user oriented application is without any doubt some form of visual dialog window. Using a window with all its components (i.e. menu, buttons, lists, etc.) the developers are able to model the control of any kind of process.

However this approach works well for most of the cases, it fails in the case of inability of using any kind of visual informational channel. Not just that in this case we can not use any windows, we can not even display any other information.

Therefore the design of the behavior of man-machine interface had to take a completely different approach. It relies heavily on the fact, that all the RAMPE behavior can be described by a simple finite state machine.

It acts upon the user actions (pressed keys) and internal events (messages from subsystems) and depending on the state it is actually in, it does the required action.

The design of such a machine however had to be done and tested by a third-party designing group.

Instead of dialog windows, the interface can be modelled by different stages, that the application assumes. Different stages have different goals and different actions can be performed. The stages can be described as follows:

Stop points survey - at this point the interface provides only informations about available stops (hosts to connect to - in RAMPE, they are called "bornes"). At first state only an information about the number of the available stops is provided, later (after users demand) the browsing of detailed list is allowed.

Navigation to the stop follows the survey. The user had now chosen one stop and the application had made the connection. PDA is now connected to "Borne" and the user can make the stop play sounds. This allows the blind person to get an idea of the stops whereabouts. User can also decide to stay at this stage (if he already knows the stop) and is already presented with actual urgent messages, broadcasted from the Borne (e.g. bus arrivals, departs, delays, etc.)

Stop data browsing - As the user is connected and physicaly present at the stop, the application can provide him all the detailed informations (those, that are usually available to visually non-impaired person at a first glance - type of stop, available lines, timetables, etc.). Browsing through the informations is also separated in more than one level (is modeled by different states).

All the stages consist of multiple states. The most important ones (those, that are vital for description of behavior) are described further.

4.2.1 Discovery of the stops

This is a default initial state of RAMPE. In here the user is not presented with anything else, but an periodic information (in form of a short sound) of how many stops are available around. The precise number is not given, yet only three states are distinguished - no stops, one stop and multiple stops.

In case of a further interest, the user (by pressing a key) may get further informations in the next state.

Pattern of behavior is described in the diagram 4.3.

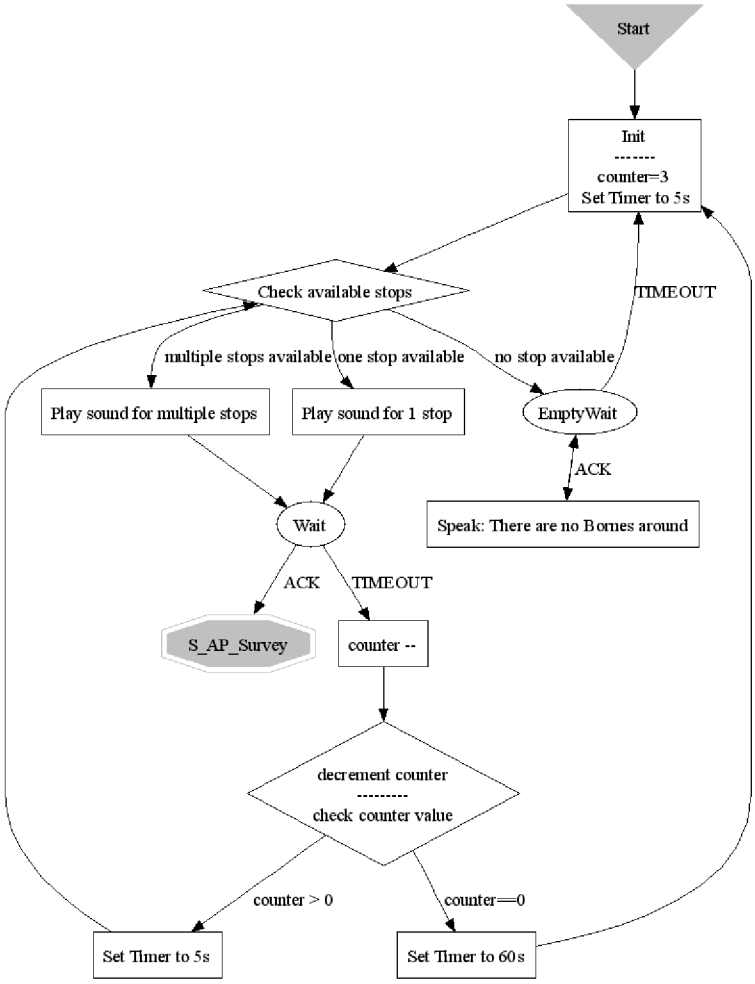


Figure 4.3: State: Discovery of the stops

4.2.2 Stops survey

Right after the user acknowledges his/her interest in more information about the available stops, this state is assumed.

In here the vocal synthesis comes to play its part. The stops are presented to the user one by one by the name and direction periodicaly. This is done several times (defined by the configuration) and the user either acknowledges one of the stops as the one, that he has an imminent interest in, or doesnt do so and then the previous state is assumed.

The graphical description is given on picture 4.4:

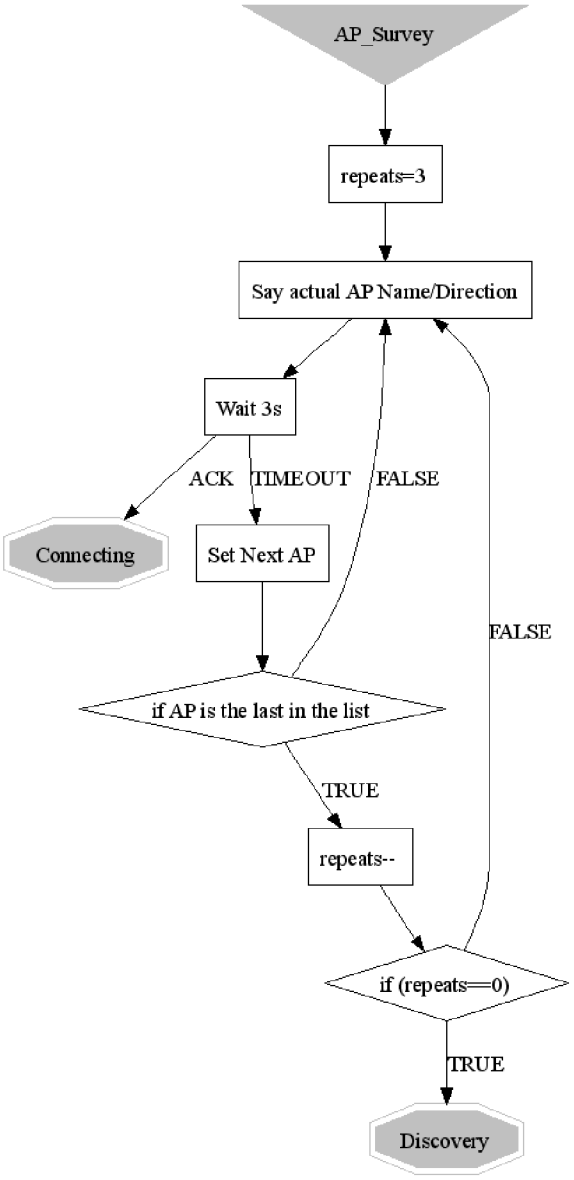


Figure 4.4: State: Stops survey

4.2.3 Guidance

If the connection to the desired stop is successful, the state of guidance is assumed.

Here the user can either demand the Borne to play the sound (to reveal its position), wait for messages coming from the Borne, or go further to the detailed informations - all of that is described at the picture 4.5.

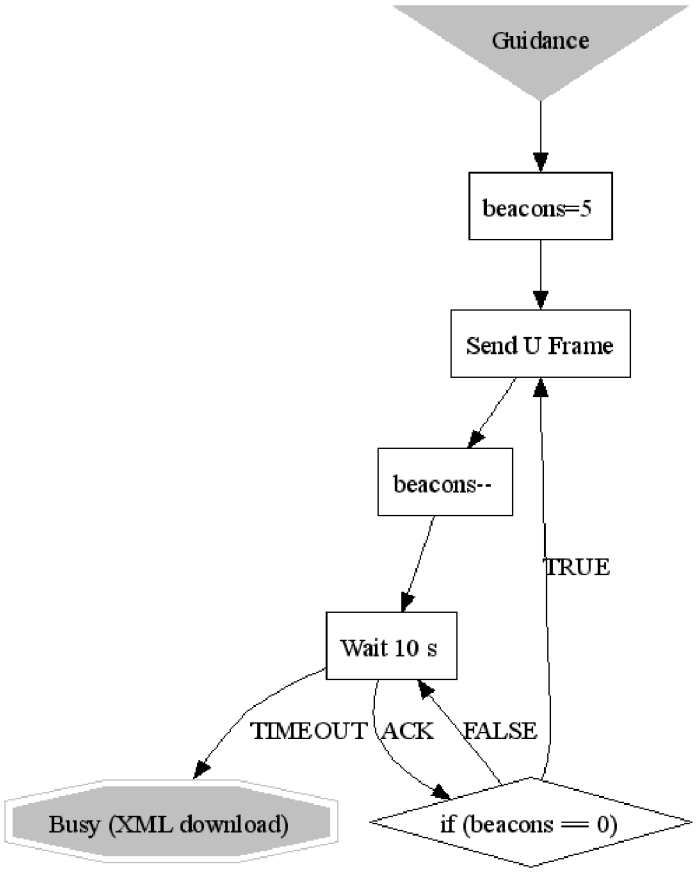


Figure 4.5: State: Guidance

4.2.4 Start of navigation

As the user has expressed a wish to be presented with more informations, the navigation through the main data file is started.

At the beginning only the presentation of the name of the stops is done. As the Borne data provide more informations (type of stop, presence of other blind persons, etc.), presentation of such informations is likely to be done here.

Proceeding from this state is automatic and no user action is required - therefore no diagram is given.

4.2.5 Navigation through available lines

As at one stop there may be several transportation lines available, a process similar to the survey of stops is started - with the difference, that not stops, but the available lines are presented (by their identifier, direction and time to wait for the next one according to the timetable).

User may either acknowledge one line to get more informations, wait until the presentation is over (it is repeated several times - again, it is configurable), or stop it and get back to the guidance.

Diagram of behavior in this state is at the picture 4.6.

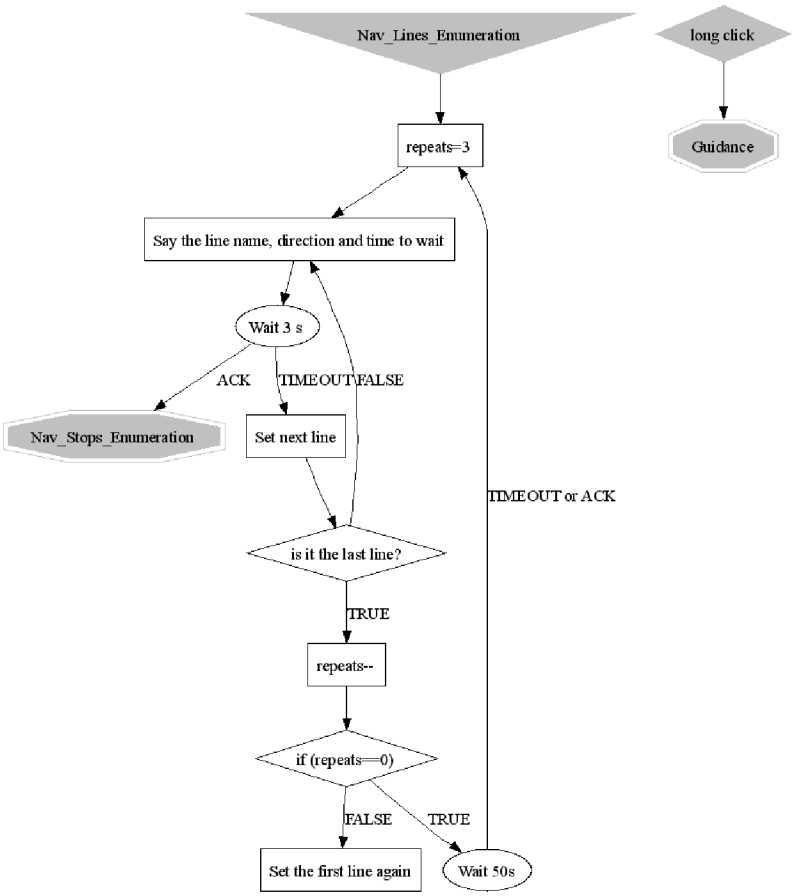


Figure 4.6: State: Navigation through lines

4.2.6 Navigation through available stops

This is so far the deepest state available. Here the presentation of the stops starts. It has two stages.

First, only the so called "skeleton" stops are presented from the current stop further. The "skeleton" stops are defined by the transportation company as somehow important (they may have intersections, they are near some popular place, etc.).

If the user acknowledges, the mode is switched to presentation of all the stops.

Diagram is given at the picture 4.7.

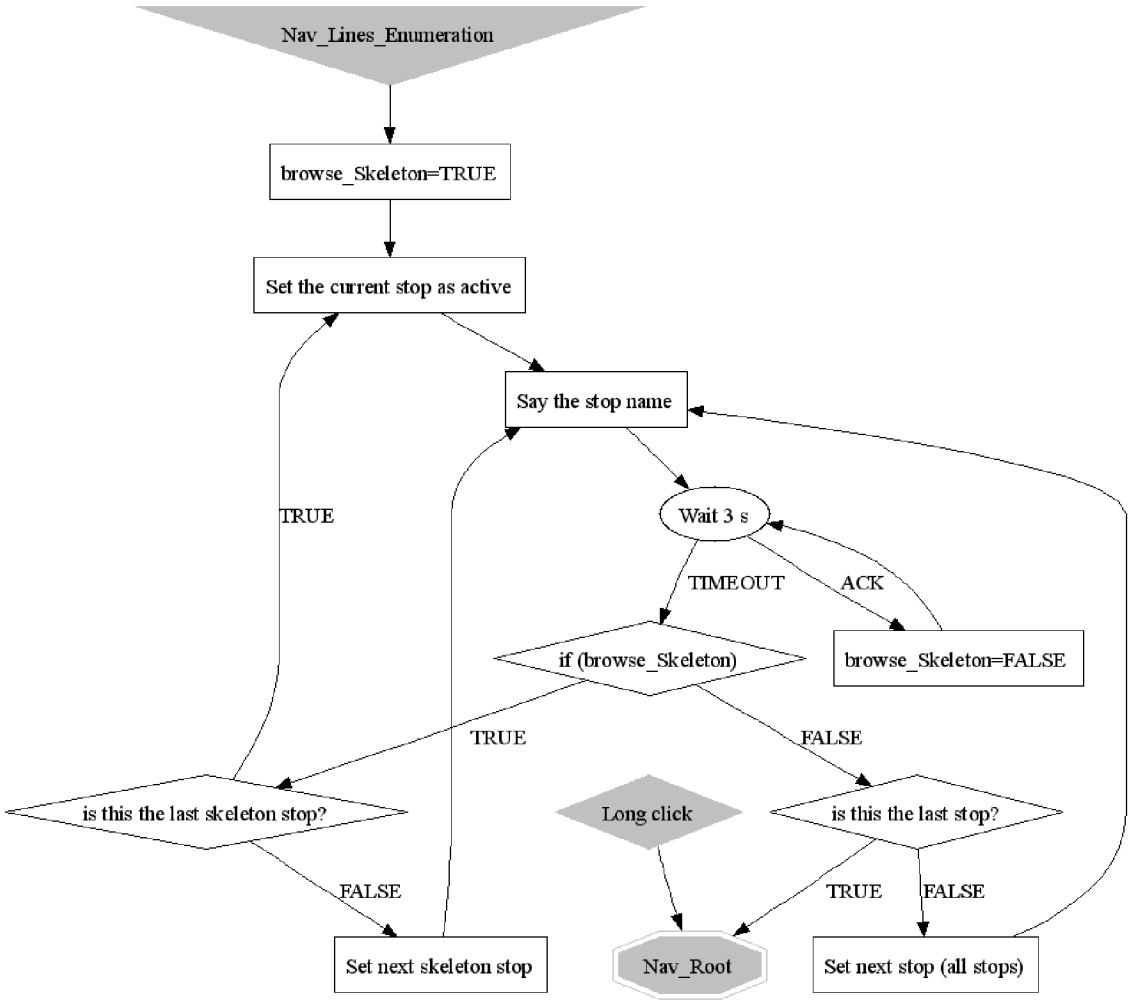


Figure 4.7: State: Navigation through stops

4.3 Processing of XML

XML is a very important part of RAMPE. Most of the data (not related to user actions) is acquired into the running application in this format. So far it means a need to process 3 types of files :

data acquired from Borne - the application gets the data from Borne by downloading the file `rampe.xml`. In this file there is everything, that can be presented to the user (bus timetables, informations about stop, etc.)

configuration data - all the configuration of RAMPE is stored locally in the file `Config.xml`. Anything, that affects the behavior of the application and can be changed is placed here.

language dependent data - although for now, RAMPE is developed only for francophone environment, this may change in future. Therefore all the messages, that are presented to the user (via TTS) are stored in a local file `Lang-XX.xml` (where XX is the name of language) and can be easily changed by third party.

As for now, XML processing in RAMPE is based on MS XML Parser, running in DOM mode. Since working with MS XML parser directly may prove to be tricky for inexperienced users, there is a wrapping mechanism built around that. That mechanism takes care of all possible issues (i.e. memory management, initialization, loading of the data, etc.) and provides the user only those functions, that are necessary for successful processing and traversing through the XML tree.

Processing of each particular file then becomes only a straight-forward "mechanical" work, not likely to introduce memory leaks or exceptions - exactly in accord with the goals specified above.

4.4 Text to speech synthetizer (TTS)

As the RAMPE project is supposed to provide informations to blind people using widely-available equipment (PDAs, smartphones, etc.), it is unlikely to provide other output than sound (since special equipment may not be available).

For the quality and easiness of use, it has been decided (in the previous phases of project) to use a commercial Text-to-Speech synthetizer, provided by the company "Acapela group" (enterprise created by fusion of three former companies - Elan speech, Infovox and Babel Technologies).

The qualities of the product in terms of functionality and provided user-experience are excellent, but there is one major flaw - licensing. Since RAMPE is meant to be widely available (possibly for free), this might be an issue (is discussed later).

4.4.1 Integration of the TTS into RAMPE

For RAMPE to keep the priorities (mentioned above), building a general TTS module (as a wrapper around Acapela API) was necessary.

Also although Acapela software has a very well designed interface, there is a good reason not to use it directly. Since it is planned for the future to switch to (or just being able to use) some other (possibly free) speech synthetizer, it would be unwise to bind the application too tightly with only a commercial API.

4.5 Network (low level) controller

Networking is the crucial part of RAMPE. Without the ability to connect to some sort of network, there is not much sense in running the application (at least at the current state of project).

4.5.1 Network adapters in RAMPE

As for now there is only one type of network adapter available and used (WiFi), but in the future developement, this will be one of the first things to change. Therefore the design of architecture had to be very careful and thoroughful.

The whole interface of a general adapter (in this case called `CNetworkAdapter`) is concentrated into one class. Such a class provides interface (but only interface, via its virtual members) for each and every function, that possibly may be required. Some of its member functions are purely virtual, since they calls are considered to be necessary for every adapter (e.g. enabling, disabling, connecting, etc.), while others are kept only as virtual, since some functions may be optional and possible to realize only at special hardware (e.g. list of available hosts is likely to be used by WiFi or Bluetooth, but not by Ethernet or USB cable).

Distinguishing between different types of interfaces is possible via provided data structure, but because there is no behavior defined in the specifications for such a scenario, the implementation of such behavior is left to be done in future.

4.6 Network (high level) services controller

There is a need for the RAMPE to be able to exchange data with the Borne (other than the main XML file).

So far the communication is based on the special protocol, created for this purpose over normal TCP/UDP. Development and testing of such a protocol is again done in different project (done by Mr. A. Sirk so far), therefore will be mentioned only briefly.

The protocol at the moment uses a simple ASCII encoding to transfer a data structure called FRAME. There are generally 3 types of FRAMEs:

Type U TCP connections from PDA to Borne, in order to make the Borne to play the "ring" sound, necessary for the blind person to be able to locate the position of the Borne.

Type V UDP datagram, broadcasted from Borne to all the PDAs in the range. It carries inside some "urgent message" (e.g. informations about a bus arrival / departure, delay, etc.).

Type R UDP datagram, broadcasted from Borne as a information for its client, saying that the XML data in the Borne have changed and the clients should refresh their informations.

The underlying implementation is done at two places - class `CRampePacket` creates / parses the FRAME structure, while `CRampeSocket` is a wrapper around WINSOCK socket.

As the connections are inbound as well as outbound, RAMPE needs to behave as a server, as well as a client. Therefore the "network services manager" had been created,

allowing the application to manage its connections and providing the higher level of abstraction.

Chapter 5

Supporting parts

In addition to the parts, that are apparent at the first glance, there are some hidden, yet still important parts, that are worth mentioning.

5.1 Parallelism and multithreading

In this kind of project there is always need for multiple processes (threads) to run at the same time.

In the code taken over from the earlier phases one could easily see quite extensive usage of multithreading (necessary for keeping the delays of user actions reasonably low).

The original implementation however had its drawbacks. Mainly it was uncoordinated spawning of the threads in different parts of code, which made the code a bit harder to understand and therefore prone to various kind of mistakes, either caused by possible false assumption of which code is part of which thread, or simply unconsciously breaking synchronization or sharing .

5.1.1 Problems

While designing a new mechanism for multithreading in the project, the biggest problem has been found in the MFC.

In the MFC, there are two types of threads (called *worker* and *UI* thread). Difference between them is, that the worker thread is supposed to be spawned for doing one time-consuming action and then returning almost the same way as a regular function. It is just executed on background in different thread from the main application. UI thread on the other hand is supposed to spend most of its time waiting for user actions (passed into the thread as messages) and reacting accordingly.

Although both seem to be sufficient for deployment "out of the box", they both have common pain - they need to be called upon global or static functions. UI threads even require hard-coded definitions of accepted messages. Those limitations are OK for one-time generic solution (such as the one, that had been used in the past), but effectively prevents the whole mechanisms from being used directly in a class hierarchy.

In the refactored code, there have been steps taken to provide secure parallel environment where necessary, while keeping the resulting code as simple (and close to the non-parallel programing style) as possible

5.1.2 Implemented solution

While designing, generic MFC calls had to be wrapped inside (dynamicaly created) classes to allow them to fit inside the designed class hierarchy. Doing so also allowed further improvements in the messaging system (namely adding the possibility of using any type as a message for a thread).

Both original types of threads were preserved, while some additional features had been added.

CThread replaces UI thread. It is able to send and receive messages (of user-defined type) and ensures, that inside the inherited class the messages will be processed sequentially - therefore allowing the user to write a classical linear code, without having to care about its parallel execution.

CWorkerThread replaces worker thread. It can be used in the same fashion, as the original (only with an obligation to inherit it as class, rather than call it as a function), when by overriding provided virtual function `Execute()` it behaves the same way as the original implementation. There is also another option - although it does not use its own messaging system for accepting messages, it is still able to send - therefore a message about successful finish or failure can be sent.

Since mandatory hard-coded definitions of messages had been taken as undesirable, new mechanism had been created to replace it. The design of all the classes for multithreading is based on templates, thus allowing the user to send any arbitrary message type (that has the semantics of a value and can be fitted into STL containers).

Whole mechanism is reasonably simple (about 500 lines of commented code) and easy to understand (and use). This is a must under the presumptions, given at the beginning.

Also by hidding the actual details of parallelism inside the implementation, it simplifies the parallel-based environment into the linear execution, thus considerably lowering the requirements to the future developers.

5.2 Data distribution inside the project

Since there is a great deal of informations, taken from outside (mostly XML files), there is also a need to distribute those informations around the application.

Doing so can be done in a generic way (e.g. by global variables), as it was done in the previous code. Although it is very simple, this approach however has its drawback.

High coupling - in case of high number of variables, their usage in the modules is prone to become chaotic and not easy to understand. Modules might then depend on variables defined in different modules, which adds additional level of dependency into the code.

Constant incorrectness - since (if not wrapped by a careful abstraction - case of the previous code) all the data have to be accessible for writing at one point (at least while parsing the data files), it is tempting to keep them writeable all the time and possibly allowing modification (which might be sometimes desirable). This leaves the fellow developers the obligation to take care of using the data properly.

Strong naming standards - if naming standards are not established and kept strictly through all the process of development, reading of such a code will soon turn into searching for definitions of variables and their meanings

Being presented with such risks, it is clear, that however sharing the data through putting them into global variables is an easy solution, it is a very tricky one too (specially under the premises given in the Goals and Priorities specification).

5.2.1 Design of data structures

The shared data (regardless of the type - language, configuration, downloaded XML, etc.) do always fit into following criteria:

- if it is not just one stand-alone variable, it is possible to structure it into a tree.
- most of the access will be read-only (although write is sometimes also necessary, it will be done far less, than reading)
- they allways will have (or it will be possible to give them) a semantic of value (given by the nature of what is about to be shared)

Those criteria give us some clue about the problems - they can be splitted into two types:

- Creating a reliable structure for storing data
- Creating an abstraction for protecting data

It can be clearly seen, that an ideal structure for storing such a kind of data would be either structured type, or STL containers (in case of a need for more dynamic behavior). However the data may be stored in different structures, the protecting mechanism may stay the same.

The general encapsulating class had been created, that contains the write-accessible data. For protection, the C++ `const` mechanism had been chosen. By giving the user a default access, that returns constants, we can prevent a unintentional change, while still keeping the possibility to get write access explicitly (by enforcing the user to explicitly ask for permission to write we can safely suppose, that the user is aware of what is he doing).

5.3 Logging facility

As the project is in the development, there is a vital need for a powerfull logging service. Such a service is provided.

In the PDA, there are two means of logging output - either screen, or a logfile. Both may be used under different circumstances for different types of messages. Therefore any call for logging a text may be assigned with a priority (or type, if preferred).

This not only reduces the amount of information outputted to the screen for purpose of supervision (e.g. while training the user, or testing the application), but since each priority might define its own prefix (making it easier to find by means ouf automated processing), it also improves readability of the resulting logfile.

Chapter 6

Open issues

As the code is still in the beta stage, there are still some features (and enhancements) missing. A brief list of such "TODOs" follows.

6.1 Processing of errors

This problem is closely related to the nature of the project (and its code). As it was said earlier, the simplicity of the code is a very important goal. Although C++ provides exceptions as a very strong tool for handling various kinds of errors, the usage of such a tool might be tricky for unexperienced users.

To keep the design simple, the exceptions (if used) need to be kept within the module, that had thrown them. So far there is no global policy on handling the exceptions (and there is not likely to be one in this phase).

For now the goal of error processing is only to report the error into the debug file, so that the developer can reveal what went wrong in the application. Some parts of the code are checked only with assertions - although these are being used only for the conditions that should never occur under normal circumstances, it would be better for the stability of the application to catch those errors with other means (e.g. exceptions).

6.2 Security

The whole design so far does not seem to take the security in account, so the following section just summarizes author's opinions and suggestions.

The RAMPE project so far doesnt neither store, nor transfer any significant data (meaning - nothing, that could cause any losses on finances or health). Therefore security is definitely not one of the top priorities. Nevertheless, as it is suggested to become a widespread network, the security should be taken in account in the future.

6.2.1 Current state

As for now, the only mechanism of security is WEP encryption of the WiFi communication between the client and the Borne. In the opinion of the author, this is insufficient. The only thing WEP is supposed to provide is privacy on the WiFi network, and even that is done in a very poor manner (it is not even close to the topic of this work - more can be found at [8]).

6.2.2 Possible targets

The network so far has two possible objects of attack - the client PDAs and the network infrastructure (Bornes and its controling mechanisms).

Network infrastructure - as is not a topic of this work, it will not be covered in-depth. Briefly said, the possible attacks may target either the availability of service (multiple types of "Denial of Service" attacks), or placing a counterfeited Borne into the network. Neither of them is covered by the specifications so far.

Clients At the client side, the attackers options are quite small. The exchange of data between Borne and PDA is so far limited only to sending XML files (may be prone to errors in XML parser) and sending RAMPE frames (may be used for misguiding the user).

6.2.3 Real possibility of attack

As it has been said - RAMPE is not a vital service and due to its nature (public service for a small minority in the society), the motivation of possible attacker is a question to ask. Also the possible impact of an attack is (at the current scenario of usage) very limited.

The security issues however will need to be resolved, if the project will be deployed in mass scale, or for general public (or commercial) usage.

Chapter 7

Conclusions

7.1 Current state of the project

Right now the whole RAMPE project is in its second phase - meaning, that it should have a working implementation and should be only extended functionaly.

Due to the steps taken (refactoring, changes in the architecture), this is true only partly. The progress in terms of functionality was only minor.

In the common terms of software release cycle, the state of project might be marked as "beta". There is no known major bug, that would cause the application to crash repeatedly or prevent the usage in most cases, yet the testing has not been finished yet and minor bugs are expected.

The original functionality had been achieved and improved. The implementation is working and tested 16^{th}-27^{th} of April, 2007 in Lyon. Written report and official conclusions from the testing are not known at the time of writing this report, but preliminary results show only minor bugs (typical for the "beta" stage),

7.2 Possible functional improvements for future

7.2.1 Extending of the configurability

As for now, the configuration relies on one locally stored XML file. Just like everywhere, the possibilities of configuration are usually endless.

7.2.2 Internationalization

However the project is developed in France and by French institutions, it is possible that more parties will be interested to contribute.

The project so far includes a separate local XML file for storing language dependent data, but the TTS software is so far available only in French.

7.2.3 Availability for different platforms

Multiplatformnes had been asked by the supervisors to be one of the major concerns, the environment chosen for development does not make it too easy.

As the mobile devices market is notoriously known for neither having a leading player, nor being well standardized, it will be difficult to assure the portability of the applications.

Also as the market is changing rapidly, it remains unsure what platform will be the best one for future.

7.2.4 Additional improvements in man-machine interface

As for now, the only way of taking user input is the keyboard of the PDA and a limited usage of gestures. This is surely enough for the application to perform its functions correctly, but it can still be improved.

Improvements may focus on following topics:

Improvements of the statemachine, using the available means. However improving the coordination between gestures on the touchscreen and keypresses may always take place, the author personaly does not see much potential in this field (or more precisely - the potential may soon reach the point, where further improvements bring only doubtful value, while introducing costs on the developers time).

Introducing new input hardware. This may likely consist of modified external keyboard, that would be easier to use for the blind person. Such an action may greatly simplify the usage of the application. Actions, that are now bound onto a combination (or a special mode) of keypresses may take bindings with only a single key.

Introducing voice recognition software. Last but not least - this option is in author's opinion the most promissing one, although it is also likely to come with the greatest cost. The voice recognition may in practice remove all the problems, that the man-machine interface may have now (e.g. ambiguity of actions). The problem is in this case likely to be the cost. Not only the implementation of such a software on given platform is likely to be more complex (due to problems with very noisy environment and low compuitation power of the machine), than the main project itself so far (no freely available alternative is known at the moment), but also the performance requirements may be too high to bear.

7.2.5 Navigation techniques

Since the users of the application are likely to have problems with orientation, while at public (or unknown) space, increasing (at least partly) their possibilities is highly desirable.

As all the application is biased towards public transportation, the navigation possibilities might be very well delimited by that fact. It does not have to use only global navigation techniques (e.g. GPS), but can also profit of a limited-range means. At the moment, there is a students project running, that is aimed to determine possibilities of localization of the person using WiFi (as this is already available). Further experiments are planned with other means too.

7.3 Licensing issues

Although it might not seem so at the first sight, RAMPE at the future phases might encounter some troubles from the legal side. Main reason for that would be licensing of its code and supporting parts.

7.3.1 External dependencies

Probably the major concern for future will be an attempt to free the project of all the unnecessary licensing fees and restrictions. For now the major issues are the usage of proprietary text-to-speech synthetizer and proprietary development platform.

Although both of these will be painful to replace, both issues will have to be settled before the project is released for public usage. If it does not happen, the burden of the licensing fees and legal problems might prove to be too difficult to cope with.

7.3.2 Internal restrictions

Problems might also arise from the cooperation of different parties inside the project. Although considerable part of work is being done by public institutions (universities and technical university), major part is still being done by a private company. This might prove to be limiting factor in the case, that the directors of the project would like to open the development model.

7.4 Related works

Currently, there is only author working on the RAMPE code itself. There are nevertheless several projects, that might be used in RAMPE development later.

7.4.1 Projects running on ESIEE

First one is a teamwork of ESIEE students, that is trying to build a database of the city environment sounds, recorded via the PDA. This should help later for possible development of a speech recognizing system.

Second one is the work of Mr. Andrej Sirk, who is at the moment working on testing and development of a custom network protocol for the usage in the RAMPE project (for communication between different devices - stops, clients, etc.).

Next one is a students internship on ESIEE, whose main object is to explore the possibilities of using WiFi as a mean of navigation. This should include exploring the possibilities of available hardware, building a test environment and measurements of the possibilities. Out of the results of this work the potential of using localization through WiFi should be determined and the next heading of the project (merging into the RAMPE) should be set.

Last one is another ESIEE internship, that aims to implement module into the RAMPE code, that could be used for controlling the Bluetooth adapter. The code is prepared for such an options and a general interface for a network adapter is implemented and used. There is however lack of agreement on how precisely should such an adapter be used. Meanwhile the possibilities of Bluetooth should be also explored.

None of those however had provided a publicly available report, therefore their results will have to be incorporated into the project later.

7.4.2 External projects

This work is being done in cooperation with external institutions. It is however out of author's competence and authority so far to interview such parties. Their complete list and roles can be found in [4]

7.5 Summary of results

7.5.1 Achieved goals

As for the goals set in the beginning, it can be said that they were (more or less) achieved.

Well structured code - although the word "well" might be a matter of debate and is very relative, the code now has an architecture and is separated into modules.

Interfaces for modules - done.

Independence of modules - there are always dependencies, but they have been minimalized.

No overcomplex design - although it had been considered to design the structure according to the established trends in modern object-oriented design, it had been relinquished. Although there are perhaps more effective design approaches (generic design, design patterns, etc.), their usage would make the code hard to understand for unexperienced programmer, hence limit the impact of the work of short-term developers.

Better functionality than the previous version

The parallel work of multiple developers - had been made possible

7.6 Contributions of this work

As the original suggested description said, this work was supposed to "improve the design", "add new functions", and possibly "add new hardware or software modules".

Out of those, the completing of the first one was more complicated, than originaly expected, therefore somehow limitting the remaining two (the improvements from the functional point of view can be said to be only minor).

The redesigning and refactoring of the original code, although painful and time consuming, was necessary and doing so had allowed adding some extra possibilities at almost no additional cost.

The main contribution therefore is improving the original flawed design and structuring the refactored code accordingly. As a side effect, the whole code had moved from the original generic mixture of C and C++ into a object-oriented pure C++.

Additionaly, the readability had been improved and the new code had been written so that it takes the ease of modification in account.

By introducing the relatively independent modules splitting the work amongst several developers became quite a simple task (as long as each developer will keep modifying just his module, while preserving its interface).

Also a general toolset had been created, that encapsulates the platform-dependent code (e.g. XML parser, network driver, threading model, etc.). This might greatly reduce the problems with possible portability of the code to different platform (however the restrictions caused by the usage of MFC and Microsoft Visual C++ compiler might still cause troubles).

Chapter 8

Appendixes

8.1 Appendix 1. Multithreading in RAMPE

Since threads are used extensively through all the application (stated in the main text), their usage had to be unified.

Since the recommended way of usage in MFC is complex and not always clear to understand, the wrapping mechanism had been used. The main idea follows the MFC. There are two types of threads (worker and UI). Each has its respective class, designed as an interface to be inherited (spawning them empty does not make much sense, although it is not explicitly prohibited).

The main ideas during the design followed the policies set in chapter "Specification and requirements". The mechanism does not try to replace all the options, provided by MFC classes. It rather provides an environment, that (although it is run as thread in a parallel with others) behaves as a normal "serial" code. If there is a need to share data with the outer world, either the implemented messaging, or provided template for sharing variable can be used.

8.1.1 Implementation and usage

Both classes (**CThread** and **CWorkerThread**) are designed as template base classes, yet both are supposed to be used in different scenarios.

CThread is a template class with a parameter representing the type of message, that the thread accepts.

Mainly it provides purely virtual function `OnMessage(CMyMessage`Msg). This function needs to be defined by the user and represents the actions done after receiving message. It also works as the main thread loop (any code that needs to be done in the parallel mode belongs here).

Messages are sent by provided function `SendMessage(CMyMessage`Msg), that is called in the execution time of the calling thread (reason for this behavior being the possibility of sendig the message into the queue even if the called thread is busy).

CWorkerThread is mainly simplification of the function above.

It doesnt accept any messages at all and provides one function to be overriden `Execute()`. This function works as a main thread loop. Optionaly it provides the

user with the function `Done()`, that may be called after the thread work is finished to send a message about the result of a performed operation.

For deciding which class to inherit, a simple "rule of thumb" can be summarized:

- If there is a large part of a code (typicaly encapsulating one whole functionality - networking, or speech synthesis for example), that needs interaction with the user or other parts of application, it is to be encapsulated in a class, that (amongst others) inherits the `CThread` class.
- If one only needs to perform one time consuming action that doesnt need any interaction, but one can not allow the main thread to be blocked by it (for example downloading and and parsing of an XML file), usage of the `CWorkerThread` should be considered.

8.1.2 Sharing data amongst the threads

For sharing variables for the outer world (other threads), there is a template class **CSerializer** provided. Every shared variale is then accessed using Get and Set method. Protection of the access (using mutexes) is then done by the template itself.

8.1.3 Using the CThread class as a general interface

Since the provided `CThread` class is a template, it may be well used as a interface to any general module (the type of module is defined by the type of message, that is passed). It is in fact used so in the main state-machine, where only the pointers to the base `CThread` class (with an appropriate parameter for the template) may be stored. While initialized by the particular implementation, the C++ polymorphism allows (and encourages) such a behavior.

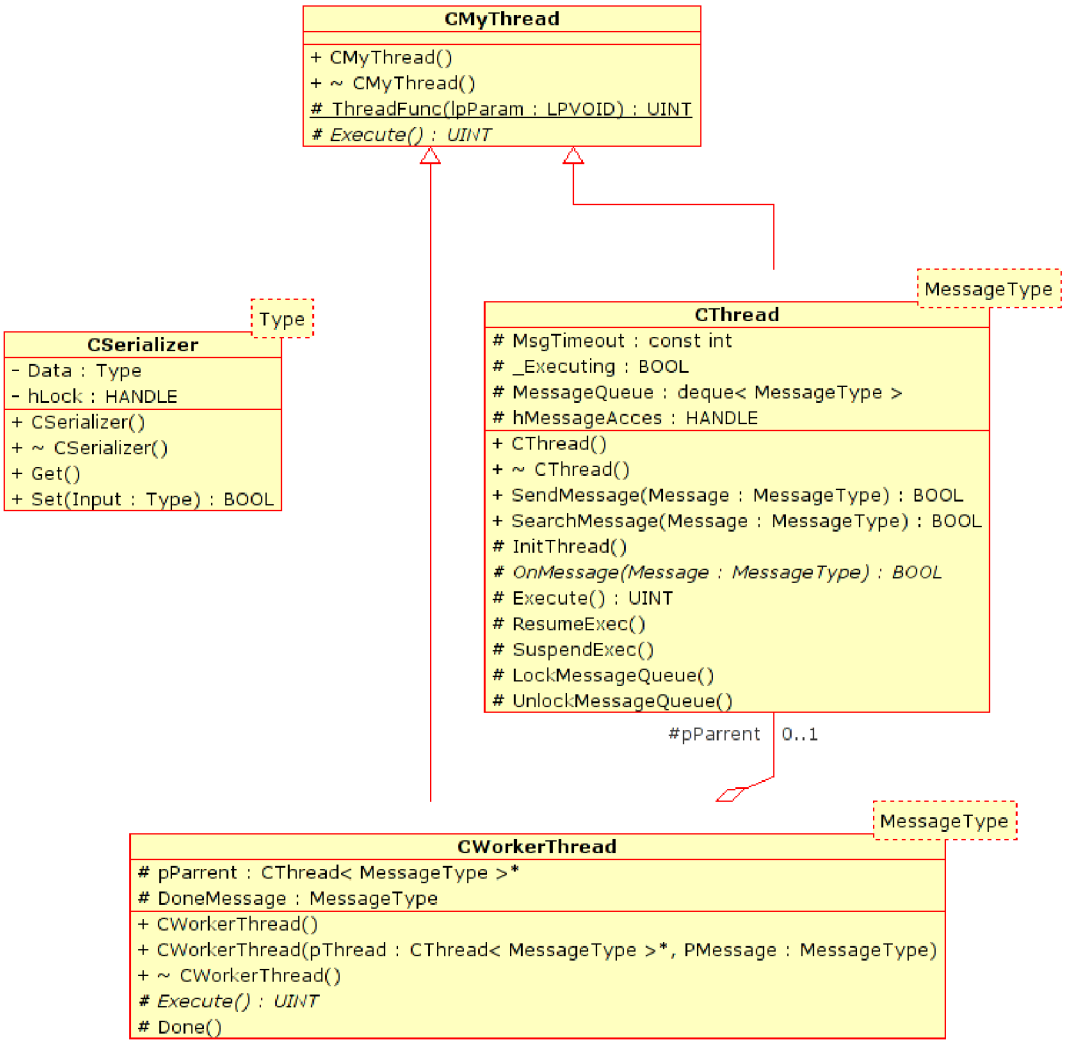


Figure 8.1: Parallel processing structures - class diagram

8.1.4 Possible improvements

The mechanism had been implemented as a sort of "naive" and easy wrapper. It does not handle more complicated issues. Some possible improvements for future (if needed) are described here.

Threads manager may be added. So far every thread is very autonomous and it's only up to the author of the code how will he handle some exceptional situations (timeouts, stalls, etc.).

The instance of general thread manager may be added, for possibilities such as checking the state of thread (running, responding, blocked) and managing errors (e.g. restart of the thread, that is blocked).

Improved messaging system Again, the messaging is only very basic. There is a queue of messages, that are processed in the same order, as they have been received.

There is no code for handling things like message priorities, or messages filtering (both can be avoided so far, but both may be useful for more complex scenarios).

Improved template for sharing The template for data sharing (`CSerializer`) has only two operations defined - `Set()` and `Get()`. That may be extended to the "value semantics" (either for usage inside STL containers, or for easier usage).

8.2 Appendix 2. Network adapter and WiFi structure

General introduction about the network adapters and WiFi usage had been given above. This appendix is therefore meant to give a bit closer insight into the implementation.

The need for a general interface for the network adapter control had demanded creating of the `CNetworkAdapter` class. It has two main functions:

Definition of base class interface , which is motivated by the demand stated above.

Definition of general behavior , which is done using the parrent `CThread` class by defining reactions for defined messages.

For future developers an example is given in the implementation of the WiFi adapter control. The easiest way to fit any new adapter (that can roughly fit into the control scheme, given in `CNetworkAdapter`) into RAMPE is simply inheriting the interface in the desired class.

Defining appropriate new behavior in the state-machine may be necessary, if such an adapter should be controlled while preserving the original WiFi possibilities (otherwise it can simply take its place without any modifications).

8.2.1 WiFi controller implementation

The WiFi controller is built upon the sample code, delivered by HP for the developers for the IPAQ product line. The code itself is not public (at least not as far, as author of this work is informed), therefore no quotation is made (except of an appropriate place in the code comments and doxygen generated documentation).

It relies upon two mechanisms - `IPAQUtil` library from HP, and NDIS API, provided by Microsoft. However the `IPAQUtil` library provides (for the WiFi adapter) some similar functions (subset, mainly for hardware control) as NDIS, it is supposed to be far more reliable (given to the nature of variations in the available hardware, those informations are easy to belive).

The main burden of functionality lies on NDIS - or, more precisely, the NDISUIO layer. The available (meaning freely available) vendor informations about that are sparse - the operations and abilities of the API is described well at [2], meanwhile the reasons for redundancy (compared with the NDIS, it brings similar functions) and the overall structure of this part seems to be (specially in the Windows Mobile part) not documented by vendor at all.

The NDIS / NDISUIO wrapping mechanism is implemented in the `CNdisuioWrapper` class. Although the NDIS mechanism is wrapped into abstraction, it still keeps the NDIS approach. There are three main methods - `SetValue()`, `QuerryValue()` and `GetLastError()`, that use a general (void pointer) parameters to set or get a specific structures. This approach, however functional and effective in practice, would not be much worth as an example, therefore HP provided another wrapper (called `CWiFiHelper`) to implement commonly used functions (e.g. connect, disconnect, scan, etc.), using the primitives of NDIS.

The functions from the provided `CWiFiHelper` class are then used in the implementation of the WiFi adapter controller.

8.2.2 Possible future improvements

The WiFi adapter controller itself should be stable and not require any future change (apart of those, resulting from adaptations to new hardware functions or software APIs).

What may be subject of change however is the general handling of the adapter in the main state-machine. So far it relies on maintaining one pointer to the base interface class, that is initialized by the particular implementation instance. If there is more controlers implemented, this provides control only over one of them at the given time.

It may be easily enhanced by adding one variable for each new type of adapter (presuming, that each will result in slightly different behavior in the state-machine), but it is real only for reasonably small amount of the adapter types (over three it may become too complex to deal with).

Such an approach may be sufficient for testing (and given the absence of the definition of behavior for the other, it is the only one possible), but in the case of a need to keep control over multiple adapters, this might prove insufficient. In such a case, a general implementation for the managing object for network adapter controlers may be required (in similar fashion, as the one implemented already for network sockets, or the one suggested for threads).

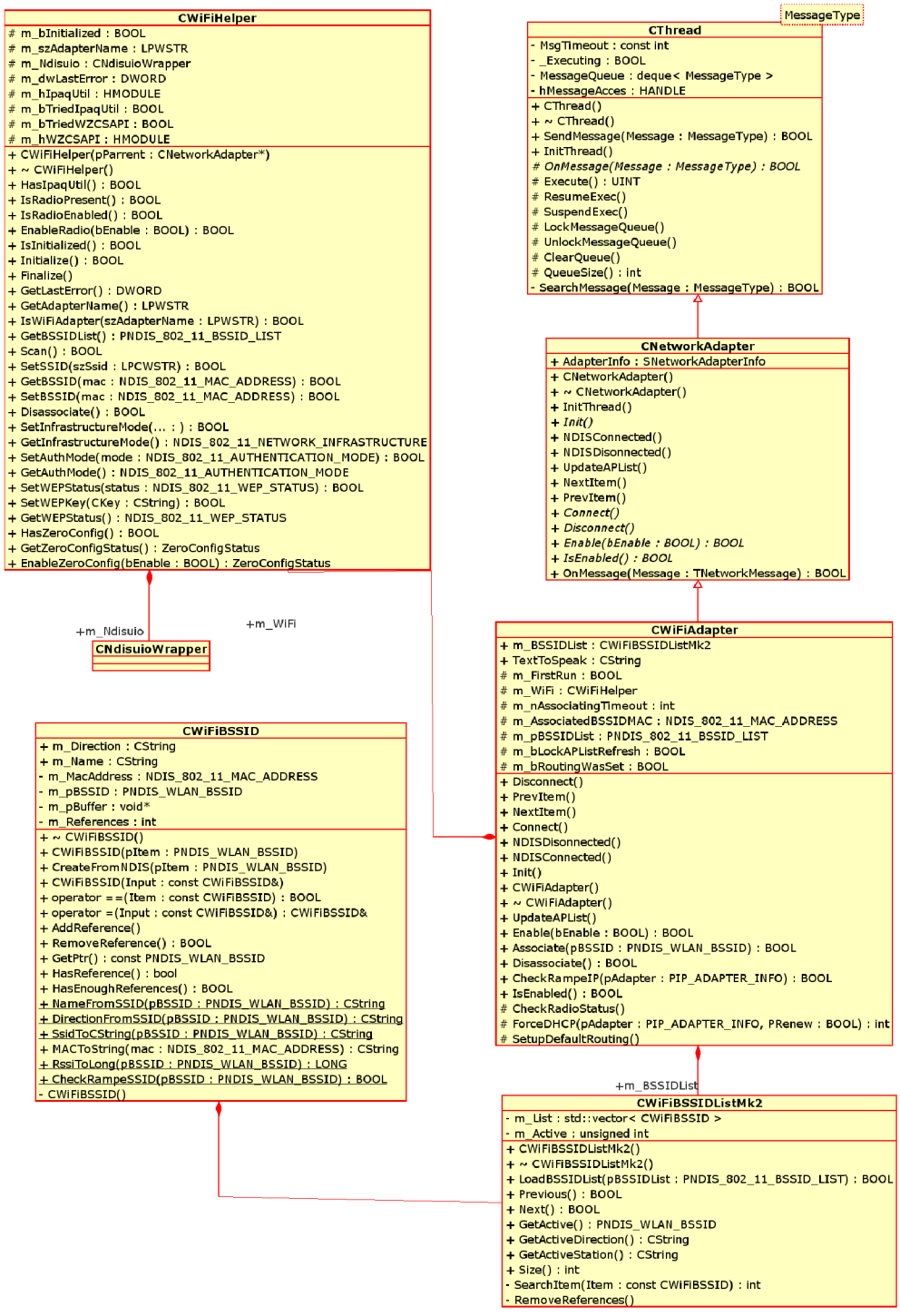


Figure 8.2: Network adapter and WiFi structure

8.3 Appendix 3. TCP/IP connections management

As the previous section handled the low-level control of network adapters (in ISO/OSI model it would be layers 1 and two - “Physical” and “Data/Link” layers), the sending of the data itself is handled by a higher-level mechanism, described here.

As the RAMPE data communication is completely realized on TCP/IP (either using HTTP protocol, or the RAMPE-defined frames), Winsock had been used for implementation of such a system.

Scheme of the position of Winsock in the ISO/OSI model is given below (as taken from the official manufacturers documentation at [2]).

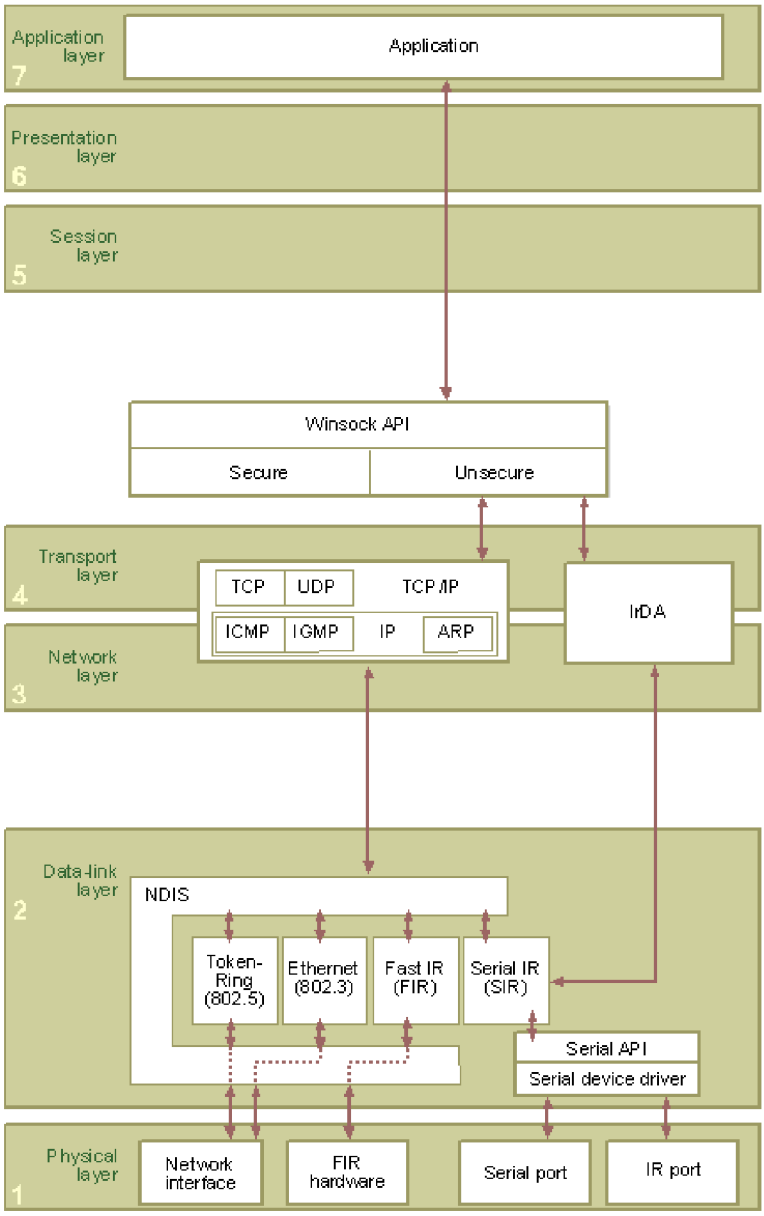


Figure 8.3: Winsock (official documentation from [2])

As the crucial structure in the Winsock is the `SOCKET`, the application encapsulates each such a resource in an appropriate object (there are only `CTCPClientSocket` and `CUDPServerSocket` defined, since those are the only ones used so far). Those object handle the management of their respective resources.

As those objects require raw data to be sent, the RAMPE frames need transformation to and from the ASCII stream. This is done inside the `CRampePacket` class.

For working with HTTP, there is another separate component in the Windows Mobile environment, therefore it is handled separately by a `CHttp` class (not described here - the techniques used there are on yet a different layer of ISO/OSI - level 7, "Application layer").

8.3.1 Possible improvements

As for the sockets infrastructure, things are working fine. The possible improvements may rely mainly on adding new sockets (missing are TCP server and UDP client) if necessary, or developing new protocol, since the ASCII frames are neither the most effective, nor most reliable way.

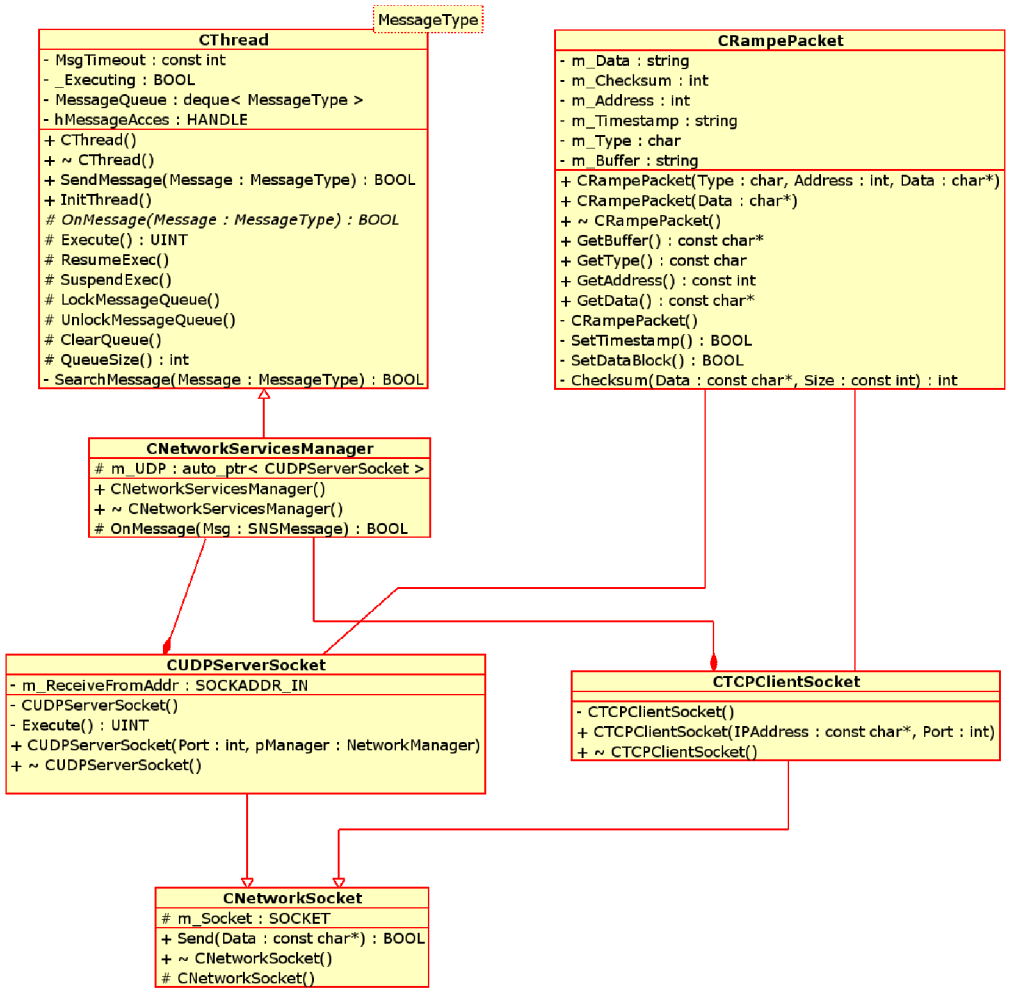


Figure 8.4: Sockets management

8.4 Appendix 4. XML Parser structure

XML is used as a main format of data transfer in RAMPE, therefore the XML parsing is used extensively through the code. Since there is plenty of available XML parsers, the implementation of the parser is relying on wrapping one engine with an abstraction, hence allowing possible future change of the engine itself.

So far the MS XML parser that is shipped with the system is used. The general parser structure relies on two main classes [10].

CXMLElement is the abstraction of one XML element node. Its functions allow all necessary operations inside the parsed XML tree (e.g. getting the values and attributes, iterating through the children nodes, basic types conversion, etc.). The main idea for such an abstraction had been taken from .

CXMLDocument is the basic class, that handles the initialization of the parser and parsing given input data. The browsed tree is saved in a form of its root element (`CXMLElement`).

While implementing a particular tool to retrieve desired data from the parsed XML tree, one needs to inherit the `CXMLDocument` class. By loading it with data the XML parsing is started and the result is stored in the `CXMLElement` object. Using the methods of that object yields the desired data, that are then processed as the author pleases.

While implementing such a tool, one may desire to separate it into different thread, for its consumption of time. This may be done easily by inheriting the appropriate class (since such a process does not usually require more interaction, than sending a message to its owner about the result of its operation, usage of the `CWorkerThread` class is recommended).

An example diagram of such a tool is given below. Note the usage of `CHttp` class. It is used for retrieving the remote file. Such an action in general (retrieving of the data) is fully up to the newly created tool (or its owner).

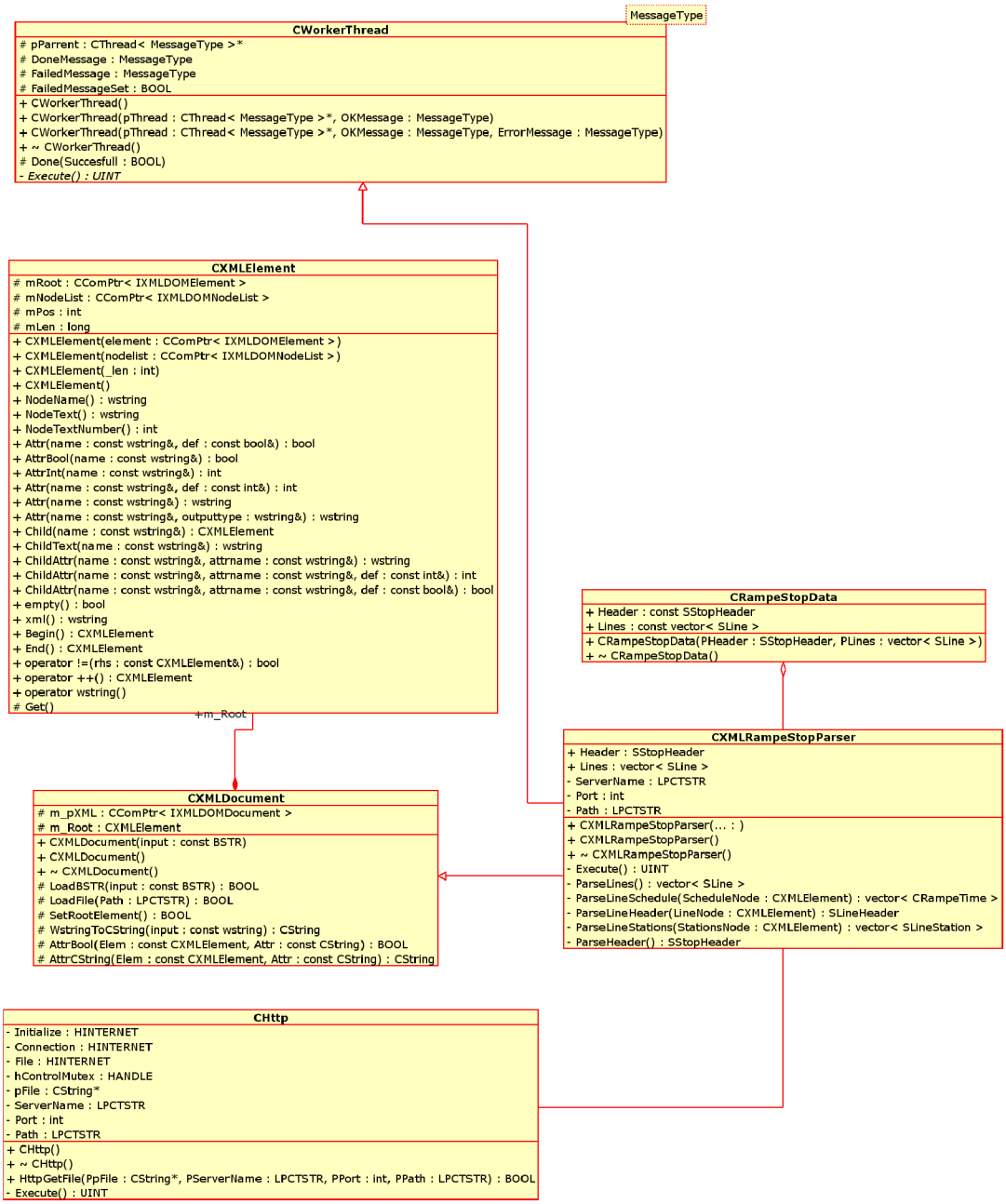


Figure 8.5: XML

8.5 Appendix 5. Examples of the XML Files used in RAMPE

8.5.1 Configuration file

This file stores those values in the application, that can be changed by user. Through this file all the configuration is made possible.

```
<?xml version=‘‘1.0’’ standalone=‘‘no’’?>
<RAMPEConfig Lang=‘‘Francais’’Version=‘‘1.0’’>
<Global>
<IPAQVersion>2410</IPAQVersion>
<AckOnRight Value=‘‘TRUE’’/>
</Global>

<Languages DefaultLang=‘‘FR’’>
<FR Path=‘‘\\RAMPE\\Lang-FR.xml’’/>
</Languages>

<Network>
<HTTPTimeout>3000</HTTPTimeout>
<WEPKey UseWEP=‘‘TRUE’’>
CC74B5BE2BD30A533385339335
</WEPKey>
<RampeSSID Prefix=‘‘RAMPE’’ Separator=‘‘/’’
  MinSignal=‘‘-10’’ MaxSignal=‘‘-90’’/>
<BSSIDList MaxReferences=‘‘5’’ UpdateInterval=‘‘5’’
 MinReferences=‘‘1’’/>
<HTTPServer Hostname=‘‘192.168.0.169’’
 Path=‘‘/rampe.xml’’ Port=‘‘80’’/>
<TCPServer IP=‘‘192.168.0.168’’/>
</Network>

<FSM>
<S_Discovery WatchdogTimeout=‘‘5’’ PauseTimeout=‘‘60’’/>
<S_AP_Survey ShortWait=‘‘2’’ AssociationTimeout=‘‘30’’
 Repetitions=‘‘3’’/>
<S_Navigation EnumerationWait=‘‘3’’ StandartWait=‘‘3’’/>
<S_Guidance Beacons=‘‘5’’ BeaconTimeout=‘‘10’’>
<!--The ‘‘beacon timeout’’ attribute is not timeout for
 the state as whole, yet only a period of time, that
 each beacon gives to the user (is not added - only
 the last one counts) -->
</S_Guidance>
</FSM>
</RAMPEConfig>
```

8.5.2 Language file

This file should theoreticaly store all the language-dependent data, that might be presented to the user. In practice hovewer only the vocal messages are made translatable. The error messages are printed only into the logfile and/or to the screen, therefore are not accessible by the (blind) user.

```
<RampeLang Version=‘‘1.0’’ Name=‘‘FR’’>
<Errors>
<No_Bornes_Available>
Pas de Bornes disponible!
</No_Bornes_Available>

<No_Bornes_Available>
Pas d’arret disponible!
</No_Bornes_Available>

<No_Bornes_Available>
Il n’y a pas d’arret ici!
</No_Bornes_Available>

<No_Bornes_Available>
Aucun arret en vue!
</No_Bornes_Available>

<Connection_Retry>
l’arret ne repond pas, je réessaye
</Connection_Retry>

<Connection_Retry>
l’arret ne répond pas, patientez
</Connection_Retry>

<Error_Back_To_Discovery>
Je recommence.
</Error_Back_To_Discovery>

<Error_Getting_XML>
les informations de l’arret sont défectueus
</Error_Getting_XML>

<Error_Getting_XML>
les informations de l’arret sont inutilisables
</Error_Getting_XML>

<Error_Getting_XML>
l’arret ne me donne pas d’informations.
</Error_Getting_XML>
```

```
<Error_No_Bus_Going_Today>
Fin de service
</Error_No_Bus_Going_Today>
</Errors>

<Messages>
<Stop_Direction_2s>
Arret %s. vers, %s
</Stop_Direction_2s>

<Wellcome_at_Rampe>
Bienvenue a RAMPE
</Wellcome_at_Rampe>

<Wellcome_at_Rampe>
Bienvenue
</Wellcome_at_Rampe>

<Goodbye>
Au revoir
</Goodbye>

<Verlaine>
<!-- message for testing the TTS  -->
A vous ces vers, de par la grace consolante.
 De vos grands yeux ou rit et pleure un reve doux...
De par votre ame, pure et toute bonne, a vous...
C es vers du fond de ma détresse violente
</Verlaine>

<XML_OK>
les informations de l'arret sont disponibles.
</XML_OK>

<Skeleton_List_Done>
Liste des arrets principaux.
</Skeleton_List_Done>

<All_Stops_List_Done>.
C'est tout.
</All_Stops_List_Done>

<All_Stops_Prefix_1s>
%s
</All_Stops_Prefix_1s>

<Skeleton_Prefix_1s>
%s
```

```
</Skeleton_Prefix_1s>
<Stop_Name_Street_2s>
%s, %s.
</Stop_Name_Street_2s>

<Line_Direction_2s>
ligne %s, vers %s
</Line_Direction_2s>

<Going_To_Discovery>
je recommence.
</Going_To_Discovery>

<Going_In_Time_1d>
<!-- message, that is appended behind the
 line number/direction. States in how many
 minutes (%d parameter) the line goes -->
dans %d minutes
</Going_In_Time_1d>

<All_Stops_List_Starting>
Je commence tout les arrets.
</All_Stops_List_Starting>

<Skeleton_List_Starting>
Je commence skeleton.
</Skeleton_List_Starting>
</Messages>

<StateNames>
<Discovery>Découverte des arrets</Discovery>
<Discovery>Recherche des bornes</Discovery>
<Guidance>Navigation vers borne</Guidance>
<AP_Survey>Découverte des arrets.</AP_Survey>
<Nav_Root>Navigation</Nav_Root>
<Nav_Lines>Navigation</Nav_Lines>
<Nav_Stops>Navigation</Nav_Stops>
<Silent_Mode>Silencio</Silent_Mode>

<State_Message_1s1d>
Vous etes a %s et vous avez %d bornes disponible.
</State_Message_1s1d>
</StateNames>
</RampeLang>
```

8.5.3 Borne informations

This file is downloaded each time the user tries to retrieve detailed informations from the stop. It includes global informations about the stop and all the lines, that belong to the stop with their appropriate details.

```
<?xml version=‘‘1.0’’ encoding=‘‘ISO-8859-1’’?>
<arret update_time=‘‘300307182847’’ valid_period=‘‘10 mois’’ etat_borne=‘‘ok’’ amenagem

  <lignes_habituelles>
    <ligne_hab desservie=‘‘oui’’ numero=‘‘9’’ direction=‘‘SAXE GAMBETTA’’ valid_time=‘‘
      <horaire_hab update_time=‘‘300307182847’’ valid_time=‘‘06 avril 2006’’ dernier_bu
        <semaine>
          <morning>6h33 7h05 7h23 7h41 8h04 8h20 8h37 8h50 9h03 9h18 9h36 9h54 10h03 10
          <afternoon>13h12 13h30 13h48 14h06 14h06 14h24 14h44 15h03 15h21 15h39 15h57
          <evening>18h07 18h30 18h53 19h24 19h58 20h28 </evening>
        </semaine>
        <WE>aucun bus</WE>
      </horaire_hab>
      <stations>
        <station nom_station=‘‘BRON HOTEL DE VILLE’’ squelette=‘‘oui’’/>
        <station nom_station=‘‘BRON SALENGRO’’ squelette=‘‘non’’/>
        <station nom_station=‘‘BRON JEAN JAURES’’ squelette=‘‘non’’>
          <correspondance numero_correspondance=‘‘25’’>SEPT CHEMINS</correspondance>
          <correspondance numero_correspondance=‘‘79’’>CHASSIEU COLLEGE</correspondance
          <correspondance numero_correspondance=‘‘28’’>LAURENT BONNEVAY</correspondance
          <correspondance numero_correspondance=‘‘34’’>CHARPENNES</correspondance>
          <correspondance numero_correspondance=‘‘9’’>BRON HOTEL DE VILLE</correspondan
        </station>
        <station nom_station=‘‘BRON LIBERATION’’ squelette=‘‘non’’/>
        <station nom_station=‘‘LUTHER KING’’ squelette=‘‘non’’/>
        <station nom_station=‘‘LES ESSARTS’’ squelette=‘‘oui’’/>
        <station nom_station=‘‘BRON JULES FERRY’’ squelette=‘‘non’’>
          <correspondance numero_correspondance=‘‘79’’>CHASSIEU COLLEGE</correspondance
          <correspondance numero_correspondance=‘‘9’’>BRON HOTEL DE VILLE</correspondan
          <correspondance numero_correspondance=‘‘38’’>CLINIQUE DU TONKIN</correspondan
        </station>
        <station nom_station=‘‘BERNARD VALLOT’’ squelette=‘‘non’’/>
        <station nom_station=‘‘PINEL-LAENNEC’’ squelette=‘‘non’’/>
        <station nom_station=‘‘A.PARE LAENNEC’’ squelette=‘‘oui’’/>
        <station nom_station=‘‘LONGEFER’’ squelette=‘‘oui’’/>
        <station nom_station=‘‘GRANGE BLANCHE’’ squelette=‘‘oui’’/>
        <station nom_station=‘‘FEUILLAT FRERES LUMIERES’’ squelette=‘‘non’’/>
        <station nom_station=‘‘PLACE AMBROISE COURTOIS’’ squelette=‘‘non’’/>
        <station nom_station=‘‘ST MAURICE’’ squelette=‘‘non’’/>
        <station nom_station=‘‘ST GERVAIS’’ squelette=‘‘non’’/>
        <station nom_station=‘‘TCHECOSLOVAQUES’’ squelette=‘‘oui’’/>
        <station nom_station=‘‘MANUFACTURE DES TABACS’’ squelette=‘‘non’’/>
```

```
        <station nom_station=‘‘GARIBALDI GAMBETTA’’ squelette=‘‘oui’’/>
        <station nom_station=‘‘ABONDANCE’’ squelette=‘‘non’’/>
        <station nom_station=‘‘SAXE-GAMBETTA’’ squelette=‘‘oui’’/>
      </stations>
    </ligne_hab>
  </lignes_habituelles>
</arret>
```

Bibliography

[1] Andrei Alexandrescu. *Modern C++ Design*. Addison-Wesley, 2001. ISBN 0-201-70431-5.

[2] Microsoft Corporation. Msdn library. `http://msdn.microsoft.com`.

[3] ESIEE. Rampe project homepage. `http://www.esiee.fr/~rampe`.

[4] O. Venard G. Baudoin. Rampe project - phase 1 final report. `http://www.esiee.fr/~rampe/`.

[5] G.Uzan G.Baudoin, O.Venard. How can blinds get information in public transports using pda? the rampe auditive man machine interface. `http://www.esiee.fr/~rampe/05-05-16-rampe-aaate.pdf`.

[6] Andrei Alexandrescu Herb Sutter. *C++ Coding Standards: 101 Rules, Guidelines, and Best Practices*. Addison-Wesley Publishing Company, 2004. ISBN-10: 0321113586.

[7] C. Marin-Lamellet. *Final Report of the BIOVAM project phase 1 (april 1999) and phase 2 (january 2003)*. PREDIT, 2003.

[8] D. Wagner N. Borisov, I. Goldberg. Security of the wep algorithm. `http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html`.

[9] various authors. C++ tips. `http://cpptips.hyperformix.com/cpptips.html`.

[10] Lucian Wischik. Using msxml to read xml documents. `http://www.codeproject.com/soap/ce_xml.asp`.