CROSS PLATFORM SOLUTION OF COMMUNICATION AND VOICE / GRAPHICAL USER INTERFACE FOR MOBILE DEVICES IN VEHICLES

Géza Németh, Géza Kiss, Bálint Tóth

Department of Telecommunications and Media Informatics
Budapest University of Technology and Economics, Budapest, Hungary
{nemeth, kgeza, toth.b}@alpha.tmit.bme.hu

ABSTRACT

Two long-term goals of our research is to develop a standardized communication interface between the mobile device and other onboard systems and to create a parametrical, scaleable user interface, both with voice and graphical user input/output. This paper describes the main requirements, principles and aspects of a voice/graphical user interface and of a Bluetooth based communication interface. Requirements and limitations for the implementation of speech synthesis on mobile devices will also be introduced. As a sample application of a mobile device on a vehicle an SMS-reader application will be presented.

1. INTRODUCTION

The abilities of smart phones and other kinds of mobile devices were enhanced in the past few years. Mobile phones and PDAs became communication centers with the latest wireless technologies (InfraRed, Bluetooth, WiFi, GPRS/EDGE/UMTS). In addition they also possess numerous favorable features, like their increasing display panel, which allows the development of informative and intuitive user interfaces, and their quite satisfactory performance, which enables the implementation of complex calculations like speech synthesis or recognition, and other advanced tasks. Mobile devices in the automotive domain still have less importance, although their advantages could also be used both in intra-car and inter-car communication. In both cases an intuitive user interface is necessary for Human-Computer Interaction (HCI). In the automotive domain it is beneficial and helpful to use speech output and input for basic tasks (Speech User Interface, SUI) complemented with a graphical user interface (GUI) for more complex functions.

A vehicle may be the context or active partaker of several kinds of communication: facilitator of personal conversations, entertainment, it can receive safety/emergency messages from another car or some authority, give navigation information, diagnostics messages, or parts of a multimedia car manual, etc.

Choosing standard mobile devices as HCI interfaces seems to be the right choice for several reasons. The phone is already a very personal device and a unique identification and billing tool, with a lot of personal information already stored (e.g. names, contacts, calendars). It is owned and carried about by the majority of people, which they also plug into the car's audio system upon entering. Therefore it is well suited to store data for the car user interface also, which is useful because users normally do not like to spend time changing car settings and more than one person may use the car regularly. In addition, a vehicle may be sold to another owner several times during its lifetime. Items of the user interface are language/country dependent, which setting is likely to be the same as that set on the mobile.

The hardware limitations of mobile phones are quickly being lifted, as phones with faster CPUs, more memory and more advanced displays/speakers come out. These devices are also more reliable than the average consumer product.

It is also easier to integrate other channels of information using mobile phones, e.g. the notifications of a home alarm system. The board computer may also give information to the phone which can help to reduce cognitive load on the driver, so that an incoming call or an SMS signal will not distract him/her during emergency breaking or a quick steering-movement. Such cases are of increasing importance as a consequence of more and more information systems packed in cars.

An argument against such use of mobile phones may be that this way car manufacturers cannot keep the interface in their own hands, sell less extras with the cars. Still, they could award 'Certified for (brand)' labels to phones that match their criteria (or different levels of criteria for different types of certificates) and proved to cooperate well with their automobiles, thus giving them a way of controlling the development of such interfaces, but leaving to the customer to buy a device with his/her preferred price/performance ratio and to change it as (s)he sees fit. Security issues may also arise, but these would need to be taken care of by car makers anyway. It also means that some engineering effort can be spared on behalf of the car manufacturer, although some already implemented hardware components would be abandoned (e.g. car phone design).

Obviously it is not possible that all new cars come out with such phone-interface integration possibility at once, as we can see on the example of navigation systems, which have existed for over 15 years but they are only becoming wide-spread nowadays. The direction of progress to follow is still to be decided. Several projects work on establishing specifications for telematics, i.e. (in the newer sense of the word) automation in automobiles using software or hardware components. There are several in-car software platforms, including the QNX Neutrino real time operating system [1], Microsoft's Windows CE based mobile platform called "Windows Mobile for Automotive" [2], different real-time Linux variants[3], etc. AMI-C specifications [4] define a uniform set of application programming interfaces (APIs) that enable software developers to write applications that can operate in vehicles. The hardware at present is usually some embedded system, such as the Xilinx PLDs (Programmable Logic Devices) [5].

In section 2 we discuss the possibilities of creating a general interface between the vehicle's board computer and the mobile phone. In section 3, we look at issues concerning the text-to-speech part of the SUI. In section 4, we demonstrate the described concept by depicting an existing mobile phone application developed mainly for use in cars.

2. GENERAL INTERFACE FOR MOBILE DEVICES IN VEHICLES

To realize a personalized standardized communication interface in the automotive domain, speech and graphical user interfaces are required. From the automotive point of view there must be a standardized communication interface with different service classes and security policies. From the mobile device point of view the same communication interface should be integrated, and standardized speech and graphical user interfaces are required. Speech input and output are basic aspects of human machine interaction in cars, as it is dangerous and forbidden in many countries to control by hands and supervise by eyes personal communication devices while driving.

The aim of the present paper is to investigate both sides in order to define what is required to develop and implement such a system.

2.1. Automotive domain

In the car the following four main points should be considered:

2.1.1. Sensors and actuators

Sensors measure the actual value of a parameter. The parameter can be boolean (e.g. back seat is leaned) or integer (e.g. volume, temperature, maximum speed) type. Actuators react to human interactions, for example set the preferred position for the driver's seat.

There can be two ways for the (preferably wireless) input and output communication of sensors and actuators with the user interface:

a) All the sensors and actuators are wireless (e.g. Bluetooth capable) and they communicate directly with the user interface system. In the case of this solution less wire is needed, although for power supply at least one wire is still required. Furthermore, some systems (e.g. Bluetooth 1.1) employ serial communication, consequently the personalized communication device should connect sequentially to all the sensors and actuators, and in addition this solution is expensive.

b) A much better solution is when all or most sensors and actuators are connected via wires to a wireless control center, which communicates with the personalized user interface device. The centralized control of sensors and actuators makes the system extendable, as it will be described in 2.1.4. The major disadvantage is that more wires and a control center are required.

2.1.2. Service Classes

In order to make the system scaleable and usable on different types of cars with different features, service classes should be implemented. Basically service classes are divided into two categories: input and output services. This separation is required, as input and output can be realized with different methods (e.g. input with the buttons on the steering wheel and speech output). Furthermore, there are different types of services classes, like temperature, seats, Hi-Fi, speed, etc. Service classes have also sub-classes (e.g. seats service class has three subclasses: front right, front left, back seat). It is defined in every car which service classes are supported so as to know what kind of functions you can control using the personal communication interface.

2.1.3. Security Policy

Different security levels must be defined to prevent the users from reaching service classes in different situations. Basically we have to distinguish three situations: car is not in use, startup and driving. At least one security level must be defined for all the service classes either to allow or to ban the user from reaching it. For example the driver's seat position can be changed while the car is not in use or during startup, but should not be changed while the user is driving.

Service classes and the rules of security policies may be included in an XML description file. This way new service classes and different security policies can be defined during software update.

2.1.4. Standardized Communication
There are doubts about using wireless (e.g. Bluetooth) in cars, as the lifetime of a car series is about 15-20 years, but we cannot suppose that any given technology will exist in mobile devices so long. It is possible that in 5 years a new technology will replace for example Bluetooth, just as Bluetooth has substituted wired and infrared communication in several situations.

To solve the problem, the control center must have a standardized communication interface. The interface should be independent from the physical medium (e.g. Bluetooth). This way if a new physical level communication standard emerges, only the communication interface and the control center should be upgraded.

Sun's JINI environment [6] realizes a dynamically distributed system that makes the handling of sensors and actuators safe and simple; the Service Classes and the Security Policy can be included, and the communication interface layer can also be realized with JINI. JINI is used in the previously mentioned AMI-C environment, which can be a possible solution for standardized in-car control centers in the future.

2.2. Mobile device domain

Mobile devices should use the same communication interface as the control center. To make the personalized communication interface widely applicable, it must be supported by many devices. Apart from communication, a standardized speech and graphical user interface is also required. Unfortunately a lot of mobile devices do not have public Software Development Kits (SDKs), but for example most Symbian OS [7] and Microsoft Windows Mobile based devices do have it. These smartphones and PDAs are widely reachable and are rather cheap. In addition the development for these devices is similar to the development on desktop computers. Additionally, these smart devices have enough computing power to calculate complex algorithms, like speech synthesis and even recognition with limited vocabulary. The devices have rather large, color display panel, which enables the development of intuitive graphical user interface.

Unfortunately even the two above mentioned mobile operating systems are not compatible with each other. In addition, new systems can also be released anytime. Consequently, standardized graphical and speech user interface is required.

2.2.1. Standardized Graphical User Interface
The user interface should be rendered in runtime according to a description file. The description file includes the user controls (combobox, checkbox, buttons, pictures, etc.), the their position and the action that is performed when the user activates them. The description file is realized in XML (eXtended Markup Language) structure. The idea is similar to the HTML (Hyper Text Markup Language), but there are three main differences:

1) The Speech User Interface is also included in the description file (*see 2.2.2.*).
2) Functions of the source code (e.g. C++, C#, JAVA) can be called if the user activates a control.
3) The user interface XML definition file can be subdivided into service class, security policy, etc. sections.

There is an existing solution for the first aspect (i.e. VoiceXML [8] , and there are also tools that realize the second feature, but these technologies are not supported by mobile devices. There are some features that are supported by Windows Mobile (e.g. ASP.NET for Mobiles), but these are not supported by Symbian OS based phones and vice versa. Furthermore the third aspect given above is a very important part of automotive-mobile control and supervision. It should be also implemented in the user interface definition file.

Let us give a simple example for the XML description of a user control in Figure 1:

```
<UserControl Name="myTextBox" Type="textbox"
ServiceClass="Temperature" Size="120px" Posx="10"
Posy="5" Input="keys" Input="voice" Output="GUI"
Output="SUI" Action="setTemperature"
Security="All">Please define the in-car
temperature</UserControl>
```

Figure 1. XML Definition file example

In this case a 120 pixel wide textbox is rendered at (10, 5), it belongs to the Temperature Service Class, the value can be set with the keys of the mobile or with voice and Security Policy allows users to set the value anytime. The actual value is represented both vocally and graphically. If the value changes, the setTemperature function is called, and the initial content of the textbox is "Please define the in-car temperature".

2.2.2. Standardized Speech User Interface
It is dangerous and in most countries it is also forbidden to control and supervise the mobile device while we are driving. To make the usage of the mobile device safe, a speech user interface is required. The input is realized by speech recognition, and the output is produced by speech synthesis.

User independent speech recognition should be used, as the training process of user dependent recognizers frustrates drivers and they can easily loose their motivation for using the speech input. The performance of the latest mobile devices is too low for continuous speech recognition, and in case of large fixed vocabularies the calculation time also dramatically increases. If the speech recognizer vocabulary at any time is not more than 150-200 words or phrases, the speed and accuracy of recognition may be satisfactory. In the automotive domain this amount may be enough, if in a well designed, intuitive dialog system the number of elements in the vocabulary can be decreased.

Speech generation provides the vocal output of the system. It should inform users about the actual values measured by the sensors, about the possible words and phrases that can be recognized, it should read information messages, etc. Besides unlimited vocabulary Text-To-Speech (TTS) which is of limited quality, specialized very high quality subsystems for well defined topics (e.g. numeric values, dates, etc. [9]) should also be implemented. Users are used to getting very high quality audio from the car speakers and they are not interested in the technical difficulties of speech generation.

Unfortunately Microsoft's SAPI (Speech Application Programming Interface) and VoiceXML are currently not supported by mobile devices, consequently, a standardized speech I/O system should be defined, which runs on all major mobile device platforms. Speech generation/synthesis engines should be recompiled for different processors (ARM, RISC, MIPS, etc.) but the speech user interface is to be realized according to the definition file, which was shortly introduced in 2.2.1. Also a subset of VoiceXML may be used in the definition file as long as VoiceXML is not supported in mobile devices.

The graphical and the speech user interface have to handle service classes and security policies. For example users are able to roll the windows, but are not allowed to control breaks with the user interface. This restriction is required because for example the noise of the environment may influence speech recognizer accuracy and it can be dangerous, if a word or phrase is misrecognized as another command. The same "error" occurs, if the user pushes wrong buttons.

3. SPEECH SYNTHESIS IN MOBILE DEVICES

As user interfaces get more and more complex, tailoring becomes necessary so as to keep the interface as simple as possible, but not too simple, i.e. giving the user easy access to all necessary functions. Speech I/O is an enabling technology for these ends.

Speech generation already has a wide range of solutions. The limitations, coming from the mobile phone environment, reduce this scale considerably. Simply put, those solutions may have a chance to be integrated into a mobile phone that have low memory requirements and need small processing power. The development in the last decade showed that good speech quality and the above-mentioned technical limits are in contradiction to each other. The need to resolve this contradiction makes it difficult to integrate speaking systems into mobile phones.

Speech synthesis is always a compromise between good speech quality and the technical demands of the application. In the case of mobile phones this compromise is much stronger than in other application environments. For reading simple predefined messages limited vocabulary solutions are sufficient, for general text reading unlimited text readers of lower perceived quality are needed. A compromise between the two is domain specific synthesis.

3.1. How to port speech synthesis on a phone platform

Developing a text-to-speech engine for mobile devices is very similar to the development of a TTS for the old operating system, DOS. It did not support TTS; there was no operating system level interface between the application and the TTS engine, like SAPI, which is present in Microsoft Windows. Up to now the situation has been the same on mobile devices: the operating system does not support TTS, SAPI is not present. Therefore every application that uses text-to-speech must contain a text-to-speech engine as a dynamic (.DLL) or as a static (.LIB) linked library. The lack of SAPI prevents developers and companies from creating speech-enabled applications because they cannot depend on the operating system providing them with a text-to-speech engine that they could use; consequently, the license of a TTS engine for mobile devices must always be bought resulting in increment of the development and of the final product costs. Furthermore, multilingual support cannot be realized well on mobile devices. Usually different languages have different TTS engines. As long as there is no standardized, operating system level speech I/O, it is impossible to add or remove a language from an application without modifying it.

Mobile devices are getting better and better, but they still have moderate performance and storage size compared to the desktop computers and servers of the day. The high quality database of text-to-speech systems is too large for many devices, and most text-to-speech software available on the market do not support mobile devices well. Building robust multilingual support in a mobile device also requires very large storage size and TTS engine(s) that support(s) all the required languages (very few languages are supported by TTS systems on mobile phones yet). Furthermore, client side applications cannot be charged according to their usage by mobile companies. Therefore, in some cases it is worth solving the text-to-speech conversion required by the client (a mobile device) in a server.

As Figure 2 shows, the mobile client sends the text to be read with some additional flags (language, quality and character of the voice, speed of reading, etc.) to the server via a data communication channel (e.g. GPRS, EDGE, UMTS, etc.). The server selects the appropriate TTS according to the language flag and feeds in the text together with the flags. The TTS generates a byte stream or a raw audio file, which is sent back to the client via the previously used data communication channel. The client buffers the incoming stream, and plays it. It is worth using some audio compression method to make the size of the byte stream as small as possible. 8000 Hz sampling frequency and 16 bit quantization are typically enough for mobile applications, as long as the device does not have a high quality speaker. This solution can be used for software upgrade and other data exchange purposes as well.

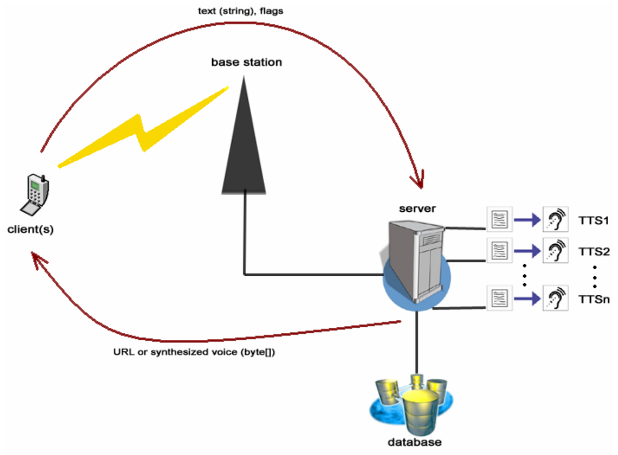

Figure 2: Client-server TTS implementation

Although a subset of SAPI 5.0 exists for Windows CE from version 4.2, there is no standard text-to-speech API for Symbian. Several companies claim to have developed TTS systems for Symbian with proprietary programming interfaces. One way to construct a TTS for Symbian is to design a TTS interface and then create dynamic link library (DLL) with this interface that will contain the TTS synthesis engine. If there is a synthesis engine implemented in standard C, there is no need to rewrite it to use Symbian-specific functions because the standard library has been implemented for Symbian as a static link library. The released application needs to have a Unique Identification (UID) number assigned to it. This number can be obtained from the Symbian Developer Network from a centrally administrated database.

3.2. Limitations and possibilities offered by phone resources

We are getting used to the fact that the storage and computing capacity of desktop computers means practically no limitation for the average applications run on them. Until recently, this was not the case with phones that are capable of running external (independently developed) applications, but the situation seems to be changing.

Phones have two different kinds of memory: internal memory for the operating system and for running applications, and possibly external memory (memory card) used for storing data only. The size of the internal memory means a kind of restriction on the program size, but the size of this memory is rapidly increasing as newer phones come out on the market. (older Symbian phones had 4 to 8 MBytes of internal memory; today one can buy phones with 64 MBytes or more of internal memory and 1 GByte or more of external memory.) The speed of phone processors (generally ARM® or Intel XScale® RISC processors) is also rapidly increasing, although it is restrained by size and low power consumption requirements.

Even though the limitations on storage and computing are beginning to be lifted, it is worthwhile optimizing programs for speed and memory consumption in order to get short response times and enabling simultaneous applications on the phone. Besides, extensive use of the processor means increased power consumption and reduced standby time. One way to achieve this is to keep as much application data in files (which are also stored in memory) as possible instead of loading them to the operative memory. This way one can avoid having them duplicated (one instance as a file and another instance in operative memory) although it can result in slightly slower operation because of the somewhat slower speed of external memory. Note that profiling the code on a desktop computer may give bad clues on what needs optimizing, as different processors execute the same code with quite different speeds.

From the viewpoint of TTS, there are three possible outputs: the speaker of the phone, the network and the headset or loudspeaker. The quality of these is also getting better, partly because of the new expectations toward such devices, like the ability to play mp3 files. 22 kHz or even higher sampling rates are already supported. Still, it is advisable to keep the TTS sampling rate at a lower rate, e.g. 8 kHz (traditional phone quality), because doubling the sampling rate practically means doubling the computation time.

4. A SAMPLE APPLICATION: SMSRAPPER

Short Message Service (SMS) is a very simple, effective and economical method of GSM communication. We cannot forget, however, that in certain situations handling of these messages becomes difficult. Today, initiating or receiving a phone call is possible without dialing the actual number (one-touch dialing, voice-dialing, automatic call reception). But we must – in all cases –

open and read through all our incoming messages word-by-word. This is especially dangerous while driving or while crossing the road in heavy traffic on foot, as we lose contact with our surroundings for seconds. Focusing at a nearby object wastes even more seconds from the recognition of an emergency. There are other situations as well when chances for reading the screen are limited.

The solution is speech synthesis (see Figure 3). The SMSRapper® application developed jointly by BME TMIT and M.I.T. Systems Ltd. in Hungary is, to our knowledge, the world's first application product that runs on Symbian phones and reads the incoming messages aloud according to the user's preferences. At the time of writing the technology is available for Hungarian, German, Polish and Spanish. It is already being used by the subscribers of T-Mobile Hungary. It is possible to use it both through the hands-free function of the phone or though a Bluetooth connection to the car audio system. The car can be adapted to a new user just by placing his/her phone inside.

'... the message is: Please go for Mike to the school, I have a meeting here.'

Figure 3. In-car application of an SMS-reader

The application has several settings associated with the phone's profiles (general, in the street, negotiations, silent, etc.) each of which has a default value appropriate for most users, with different behavior in each profile. For example, one can set how fast, how loud and how many times (s)he wants the program to read the message and which properties (s)he wants to hear (e.g. sender, date, time), separately for every mode. Voice with the associated language can be chosen for announcing messages. Language identification from the SMS text can be turned on or off. A time interval can be specified during which the phone, so as not to disturb e.g. your night's rest, does not read messages. It can be specified if SMSrapper is to be brought into the foreground when a message arrives.

When an SMS arrives, the program identifies the sender of the message based on the phone-number if it is in the phone's address book and reads his/her name; if not, it reads the number. The language of the message is made known to the user before it is read and this language is chosen for reading the text if it is available; otherwise the default language will be used.

Feedback from users of SMSrapper showed that other features are also needed for a more convenient use. When you drive your car, you may not want the device to automatically read out certain texts, e.g. highly confidential business information. Such messages usually come from certain acquaintances of yours, so you may want to put them on a "black-list" which contains the address-book entries whose messages are never to be read. At the same time, you may want that messages from other persons (maybe family members) be read out even if the phone is in silent mode, and put them on the "white-list". A further option is a "grey-list" if you want to be notified about someone's SMS but do not want it to be read aloud. Another extra feature that fits into the application's scope is reading the name of the caller before ringing starts. This way setting (and remembering) personal ringtones, or recording the person's name as a ringtone, can be spared.

5. CONCLUSION

Within the scope of this paper only a basic sketch of the possible advantages of a standardized integration of mobile devices into vehicles could be given. The authors are open to future co-operation with interested partners in order to more deeply explore this domain.

6. REFERENCES

[1] S.Ethier and R. Martin, "*Instant-on Technology for In-Car Telematics and Infortainment Systems*", http://www.qnx.com/download/download/10386/instant-on_mini-driver_whitepaper.pdf

[2] Edward Lansinger, "*Windows Mobile for Automotive: A Platform for Smart Telematics Systems*", Convergence 2004, Detroit

[3] http://www.realtimelinuxfoundation.org/

[4] Scott J. McCormick, "*AMI-C (Automotive Multimedia Interface Collaboration) Fostering Global Communication*", ITU-T Workshop on Standarization in Telecommunication for motor vehicles, ITU Headquarters, 2003.

[5] K.M. Parnell, "*Reconfigurable Vehicle*", http://www.xilinx.com/products/iq/ReconfigurableVehicle02AE-118.pdf

[6] *Jini Network Technology*, http://www.sun.com/software/jini

[7] http://www.symbian.com

[8] *Voice Extensible Markup Language, v.2.1*, http://www.w3.org/TR/voicexml21/

[9] G. Olaszy, G. Németh, "*IVR for Banking and Residential Telephone Subscribers Using Stored Messages Combined with a New Number-to-Speech Synthesis Method*", in D. Gardner-Bonneau ed., Human Factors and Interactive Voice Response Systems, Kluwer, 1999, pp. 237-255