Development Issues for Speech-Enabled Mobile Applications

Werner Kurschl, Stefan Mitsch, Rene Prokop and Johannes Schönböck
Research Center Hagenberg
Upper Austria University of Applied Sciences
Hauptstraße 117
A-4232 Hagenberg, AUSTRIA
{werner.kurschl, stefan.mitsch, rene.prokop, johannes.schoenboeck}@fh-hagenberg.at

Abstract: Developing a speech-based application for mobile devices requires work upfront, since mobile devices and speech recognition systems vary dramatically in their capabilities. While mobile devices can concisely be classified by their processing power, memory, operating system and wireless network speed it is a bit trickier for speech recognition engines. This paper presents a comprehensive approach that comprises a profound classification of speech recognition systems for mobile applications and a framework for mobile and distributed speech recognition. The framework called Gulliver speeds up the development process with multi-modal components that can be easily used in a GUI designer and with abstraction layers that support the integration of various speech recognition engines depending on the user's needs. The framework itself provides the base for a model-driven development approach.

1 Introduction

Developing speech-enabled applications for mobile devices typically requires comprehensive analysis of several speech processing engines and different architectural approaches before the desired application can be built. In addition, framework and tool support is often insufficient, which makes developing applications difficult: speech recognition systems usually provide a native low-level API or support Microsoft SAPI or Java SAPI (see [Mic] and [Sun98b]). Common to these APIs is that enhancing a graphical user interface with speech requires additional and separate speech input treatment based on a method-call-interface; thus, the developer has to manually modify the speech recognition system's state and handle textual results.

Furthermore, speech-enabled user interfaces are often hard to use, because speech as input modality is invisible and transient. It is difficult for a user to find out what can be said in a particular situation, and to remember which information was already given. Therefore, a speech-enabled user interface needs to match closely the user's expectations, which requires usability engineering to be a central part of the development process. As a result of this ongoing process the user interface will change permanently. From a developer's perspective this means that the user interface and the application's work flow needs to be easily changeable. This can be reached with two measures: tool support and loose coupling between the user interface and the application's business logic. Both measures are common for graphical user interfaces, but, as already stated before, still not met with current speech recognition APIs.

To overcome some of these issues and to speed up the development of speech-enabled mobile applications we utilize a classification guide for speech recognition systems. Section 2 describes the guide in detail; moreover, it uses the guide to classify several commercially available speech recognition systems to give an overview of the state-of-the-art. It also shows possible use cases for each classified speech recognition system and the required type of device. The analyzed speech recognition systems for mobile devices missed the following items:

- Support of both constrained and unconstrained speech recognition
- Support of a high-level, event-driven API
- Support of easily usable speech-enabled user interface components
- Support of different platforms

Thus, we present a component framework called Gulliver for distributed speech recognition with multi-modal user interface components. Gulliver supports constrained and unconstrained vocabulary on mobile devices.

2 State-of-the-Art in Speech Recognition

Speech recognition systems can roughly be classified by the criteria shown in Fig. 1.

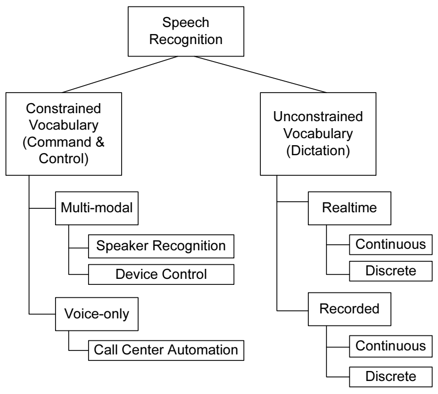


Figure 1: Classification of speech recognition systems

One of the easiest application classes are command and control applications, which typically offer a very constrained vocabulary that contains the supported commands. This kind of speech recognition is far less complex than unconstrained speech recognition and can thus be performed on mobile devices. Today's mobile phones only support voice-dialing and very simple commands; PDAs, by contrast, are able to analyze sentences that follow a predefined grammar (the grammar specifies allowed words, e.g., the numbers from 1–100). These devices can further be classified by their input modalities: on multi-modal devices, like PDAs, a user can input data via several input modalities (e.g., graphical with a pen or by voice) and controlling the device is a common use case, whereas single-modal devices like voice-only systems (e.g., calling a computer system by phone) are often used for call center automation.

Speech recognition systems that support dictation (i.e., converting freely spoken words into written text) have to provide a nearly unconstrained vocabulary. The voice input is either processed in real-time—the system provides results during speaking—or deferred processed as a batch job, when recorded speech is used (described by the speech recognition engine's processing mode). The systems can further be classified with the word model criterion: continuous systems allow the user to speak fluently, whereas on discrete systems the user has to speak single words individually (i.e., with breaks between the words). Current continuous systems already achieve high recognition rates and are, due to their convenience in use, superior to discrete systems (which have thus already become obsolete). Today's speech recognition systems with an unconstrained vocabulary are mostly speaker-dependent—which means a user has to train the system prior to using it—to achieve accurate recognition rates of 95% or more. They also demand high processing power and memory. Although the processing power of PDAs and mobile phones increases rapidly, it is far from being acceptable for performing unconstrained speech recognition locally on the device. Thus there is currently no unconstrained speech recognition engine available for those kinds of devices.

Tab. 1 shows some commercially available speech recognition systems classified by the presented criteria.

Table 1: Classification of selected commercially available speech recognition systems

	Vocabulary	**Processing Mode**	**Use Case**	**Device**
Nuance Dragon Naturally Speaking 9	Unconstrained or constrained	Realtime	Dictation	PC
Nuance Dragon Naturally Speaking Prof.	Unconstrained	Recorded	Dictation	PC
IBM ViaVoice 10.5	Unconstrained or constrained	Realtime	Dictation	PC
Loquendo Embedded ASR	Constrained	Multi-modal	Device Control	PDA
Nuance VoCon 3200	Constrained	Multi-modal	Device Control	PDA
MS Speech Server	Constrained	Voice-only	Call Center	PC

This classification reveals the current gap in speech recognition systems: depending on the type of device being used (e.g., PDA or mobile phone) only constrained speech recognition systems can be applied to mobile devices. But these types of devices are those that would benefit most from unconstrained speech recognition, as they only provide emulated or small-sized keyboards for entering text.

In section 4 we describe a component architecture that supports developing speech-enabled applications on heterogeneous devices. Its primary focus is on distributable components that enable fully featured speech recognition—constrained and unconstrained—on an arbitrary device. Subsequently section 5 describes how to ease developing speech-enabled applications with user interface components that hide speech recognition specifics from the application developer and that allow a uniform treatment of the graphical and verbal user interface.

3 Related Work

Aurora Distributed Speech Recognition (DSR, see [Pea00]) is a standard of the ETSI. DSR takes special care to minimize network traffic and thus enable speech recognition in low-bandwidth networks, like General Packet Radio Service (GPRS). So-called features are extracted out of the speech data at the client side and are transported via the network. On the server side a speech recognition engine, which must be able to recognize speech based on those extracted features, converts the features into text. Although DSR is standardized, hardly any speech recognition engine is able to recognize speech from features.

VoiceXML (specification can be found at [MBC+04]) is a markup language designed to describe dialog flows in voice applications. Comparable to HTML that is interpreted by a Web browser, VoiceXML is interpreted by a voice browser. Multimodal applications can be described using X+V (see [ACF+05]), which integrates XHTML for visual content and VoiceXML for spoken interaction. Neither standard supports dictation, because each constrains user input with grammars (at least SRGS, refer to [HM04] for the specification, must be supported by VoiceXML platforms).

Microsoft Speech API (SAPI, see [Mic]) provides a common API to deal with different recognition engines. SAPI is also available for embedded devices running Windows CE .NET 4.2 or newer (i.e., Windows Mobile 2003 or 5.0). However there are no speech recognition engines that support dictation for these devices.

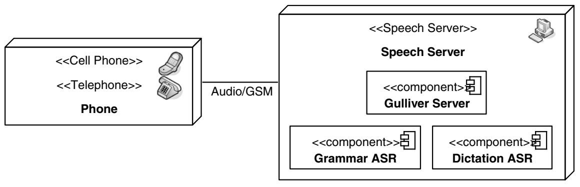
Microsoft Windows Vista [Bro06] includes speech recognition in the operating system. This means that one can control Microsoft Windows and its programs via voice commands and enter text by dictating. Application developers can easily create speech-enabled applications based on the new APIs exposed by Vista. Unfortunately, these APIs are only available for desktop PCs and not for mobile devices.

Multimodal Teresa (refer to [PS02] and [MPS03] for details) is a model-based approach used to design applications for different platforms. It generates platform-specific models (PSM) based on an abstract task model. From the PSM it is also possible to generate the user interface of the application. Although Teresa includes an editor, it is impossible to know what the user interface will look like at the end of the transformations. Furthermore the software relies on VoiceXML for describing multimodal interfaces, which does not support dictation.

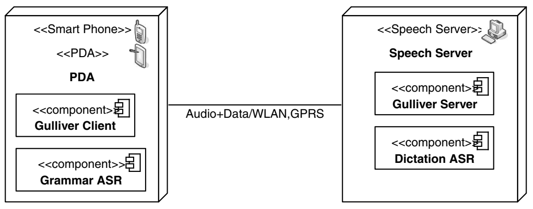
We observe a lack of functionality in different areas. Creating distributed speech applications with the Aurora standard is limited to special recognition engines, which hinders the wider adoption of speech technology in business applications. Most speech recognition engines available for building rich-client applications do not support VoiceXML directly (VoiceXML support is only available in some Web browsers). Therefore components would be needed that transform VoiceXML to a format the speech recognition engine can handle.

4 Distributed Framework

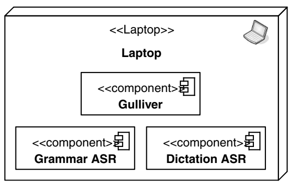
Based on our analysis described in section 2 we divided speech recognition into three different deployment schemes shown in Fig. 2.



(a) Conventional (cell) phones



(b) PDA or Smart Phone



(c) Laptop or Tablet PC (note: a separate speech server is not necessary)

Figure 2: Deployment schemes for speech recognition systems (UML deployment diagram)

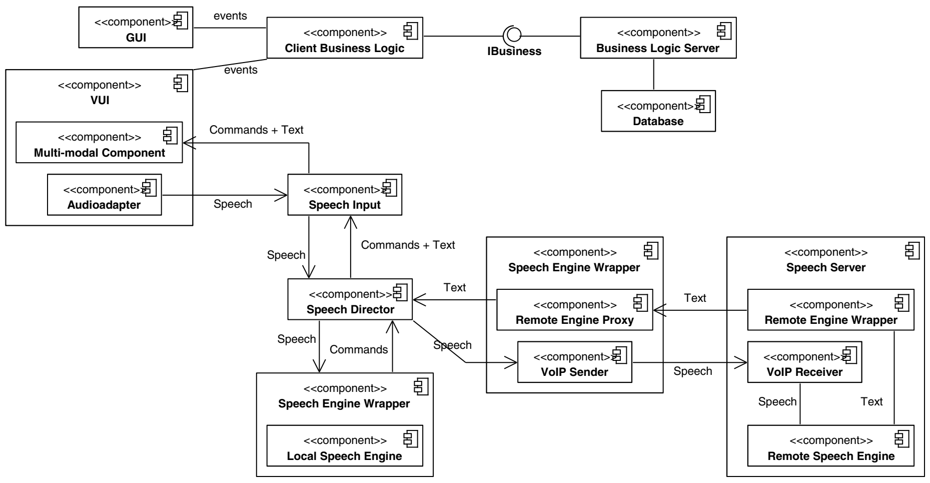
(a) Conventional (cell) phones: they are only capable of recording, digitizing and transmitting speech over a GSM or conventional telephone network. A remote device hosts a speech engine that can be accessed via phone.

(b) PDA or Smart Phone: these devices allow the client-side deployment of (multimodal) user interfaces including an engine for constrained speech recognition. But if dictation is a required feature, an additional speech recognition engine must be placed on a remote server. The framework hides this deployment environment from a developer and uses components that transfer voice data over a network (using for example VoIP) in the background.

(c) Laptop or Tablet PC: they provide enough computational power for local speech recognition, including dictation. Nonetheless a framework that separates the creation of the user interface and the business logic of an application from speech recognition eases application development.

Thus all three schemes demand for a flexible framework that hides the speech recognition from the developer by separating the speech engine from the user interface code; with current speech engines' API they are tight-knit.

We base our architecture, shown in Fig. 3, on scheme 2; in less generic scenarios some of its parts are optional.



<<component>> GUI
events
<<component>> Client Business Logic
IBusiness
<<component>> Business Logic Server
<<component>> Database
events
<<component>> VUI
<<component>> Multi-modal Component
Commands + Text
<<component>> Audioadapter
Speech
<<component>> Speech Input
Speech
Commands + Text
<<component>> Speech Director
Text
<<component>> Speech Engine Wrapper
<<component>> Remote Engine Proxy
Text
<<component>> Speech Server
<<component>> Remote Engine Wrapper
Speech
Commands
Speech
<<component>> VoIP Sender
Speech
<<component>> VoIP Receiver
<<component>> Speech Engine Wrapper
<<component>> Local Speech Engine
Speech
Text
<<component>> Remote Speech Engine

Figure 3: Framework architecture (UML component diagram)

The *Speech Input* allows the multi-modal user interface components to access the speech recognition system. They separate the development of the client business logic from the speech recognition layer. Thus, the application can use a voice user interface completely independent from the type and processing power of the mobile device. To ensure low latency for command and control tasks, a constrained speech recognition engine hosted locally on the PDA provides immediate response. To provide dictation functionality a remote engine can be used. The *Speech Director*, which implements the Message Router pattern (described in [Fow03]), controls the cooperation of the local and the remote speech engine. Hence, the number, type, and location of the engines are completely hidden from the business logic. To ensure a flexible configuration between components *Speech Channels* are used to transport voice data. Together they form a Pipes-and-Filters architecture (see [BMR$^+$96]). Depending on the scheme a Speech Channel can either be a simple stream or a VoIP component if voice data should be transferred via a network. For more detailed information please refer to [KMPS07].

5 User Interface Development

Today's frameworks for building graphical user interfaces typically use events to communicate user input to an application. This paradigm is well established and therefore familiar to most developers. Handling user input from a voice user interface should base on a similar paradigm; in most cases developers do not even want to distinguish between the different types of user input. But today's speech recognition engines only provide a generic "speech recognized" event. Thus, developers have to deal with the details of the event: often they have to examine the recognized text to decide which action to take.

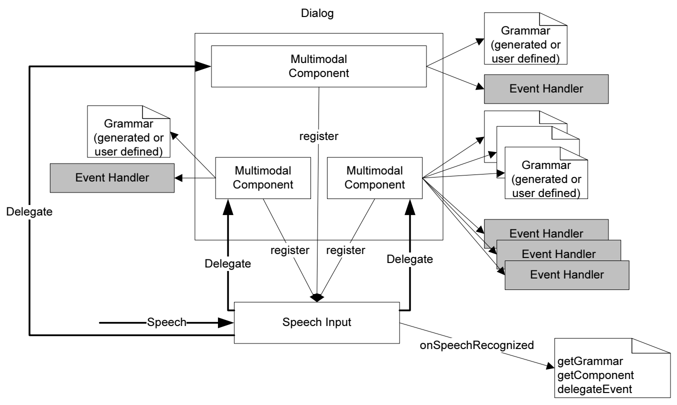


Figure 4: The Speech Input and its event handling

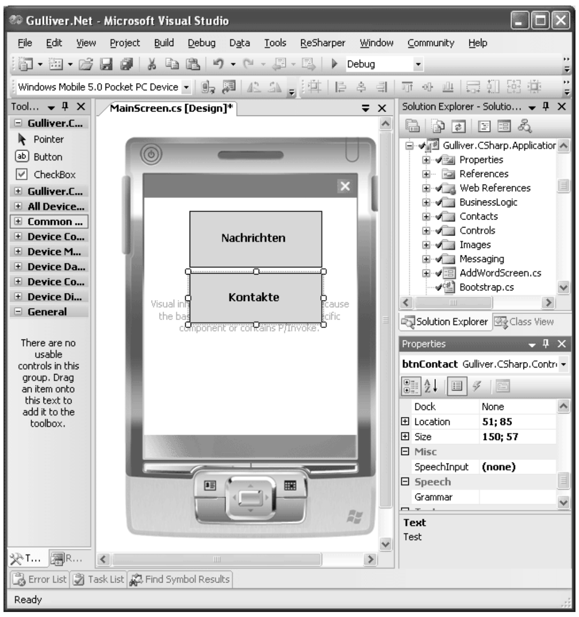
Therefore Gulliver provides components (see Fig. 4) based on top of the architecture described in section 4. These components offer two additional features compared to common graphical user interface components:

- Send events if they are operated using speech
- Allow a developer to associate a grammar or to enable dictation

Sending events after speech recognition is one main task of the Speech Input. Multimodal components are grouped in dialogs (e.g., a Windows form in C#) by the task they solve. When a dialog is activated, it registers its components at the Speech Input, which in turn activates the components' grammars in the speech recognition engines. If speech is recognized (either by a local or a remote engine) the Speech Input receives an event from the speech recognition engine. It parses this event and creates an object-oriented representation (a dictation result or grammar result) that is more convenient to use. The dictation or grammar result is then delegated to the multi-modal component. We use a Mediator (see [GHJV95]) to correlate dictation and grammar results with components; therefore each result object carries a unique name that identifies the component it belongs to. Finally, the component sends a specific event that a developer is familiar with: for example a ButtonClicked-event if a grammar of a button was recognized. The developer can handle this event in the same manner as an event from a graphical user interface component.

To handle user interfaces via voice input it must be possible to add grammars to a component or to enable dictation. Each component defines its syntax (i.e., valid input values) in a grammar. Simple components, like voice-enabled buttons, provide a generated default grammar, but they can also be customized with a developer-specified grammar. For defining a grammar there are two different standards: Speech Recognition Grammar Specification (SRGS, defined in [HM04]) and Java Speech Grammar Format (JSGF, specified in [Sun98a]). The Gulliver framework supports both standards by offering an object oriented wrapper that allows a developer to define the grammar in code using common object oriented paradigms. Each grammar object knows its representation in SRGS or in JSGF, thus abstracting from the concrete speech recognition engine's interface.

The .NET Compact Framework and Java (both are widely used for developing applications for mobile devices) provide an extension mechanism for user interface components. These so-called custom controls can be integrated in the development environment (shown in Fig. 5) and, thus, be used very conveniently.



btnContact Gulliver.CSharp.Contr

Dock	None
Location	51; 85
Size	150; 57
Misc	
SpeechInput	(none)
Speech	
Grammar	

Text
Test

Figure 5: Screenshot from an application designed with Gulliver multi-modal components

In the current version of our framework we decided to provide custom controls for the .NET Compact Framework but the underlying concepts can be applied to any programming language. Gulliver provides many of the most commonly used interface components like Button, ListBox, and ComboBox. Additional components can be easily integrated into the framework, and thus a developer can voice-enable his or her custom controls.

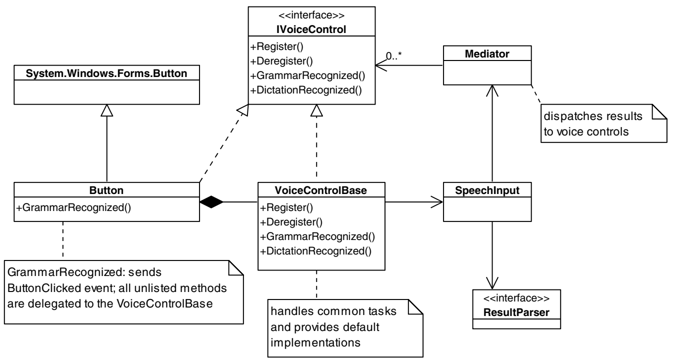


Figure 6: Implementation of a speech-enabled button as a custom control (UML class diagram)

Custom controls are integrated into the Gulliver framework by implementing the IVoice-Control interface. Fig. 6 shows the example of a voice-enabled Button. The interface IVoiceControl declares methods for registering and deregistering a component at the Speech Input—these methods are called by a speech-enabled dialog when it becomes active—and methods for handling grammar or dictation results—they are called by the Mediator to delegate results to the component. Typically, the speech-enabled custom control is derived from an existing control, thus, common tasks can not be inherited; instead, they can be delegated to a VoiceControlBase object embedded in the custom control.

Each control provides a default grammar; we suggest providing meaningful default grammars that allow the user to "say what he or she sees". In case of the button this could be the button's label. Additionally, the custom control has to define which event to send if either a dictation or a grammar result is received (which is a ButtonClicked event in our example). Consequently this means a developer just has to follow three steps to integrate new components into the Gulliver framework: *(a)* extend a control from a graphical user interface component, *(b)* implement the interface IVoiceControl *(c)* override properties/methods of the base class if they are affected by the voice input too.

6 Application Development

In this section we focus on how and where to deploy the framework's components and how to develop an application based on the Gulliver framework and the multi-modal user interface components.

Fig. 7 shows to which nodes the application's and framework's components are deployed. We assume an enterprise application (client-server or multi-tier) that performs significant business logic on a business server and hosts the presentation tier and some client-side business logic (together represented by the application in the figure) on the mobile device. The application's user interface is built with the multi-modal user interface components;

hence, it automatically uses the Gulliver framework to forward speech-related commands and grammars to an appropriate speech recognition engine and reacts to obtained results. A complete Gulliver installation comprises an optional local speech recognition engine on the device and the Gulliver Client, which is automatically deployed with the application. The Gulliver Server is a separately installed component on a powerful server and accesses a server-side speech recognition engine.

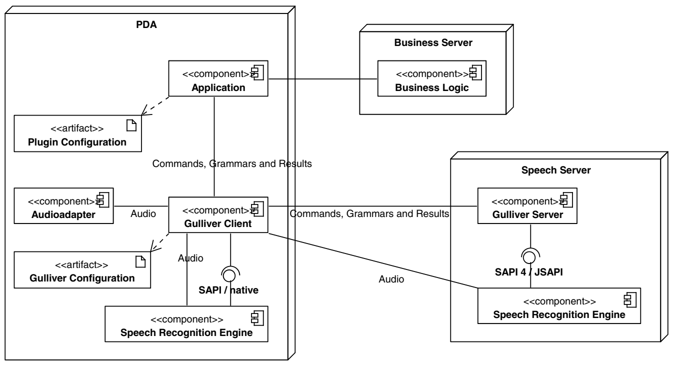

PDA
Business Server
<<component>> Application
<<component>> Business Logic
<<artifact>> Plugin Configuration
Commands, Grammars and Results
Speech Server
<<component>> Audioadapter
Audio
<<component>> Gulliver Client
Commands, Grammars and Results
<<component>> Gulliver Server
<<artifact>> Gulliver Configuration
Audio
SAPI / native
SAPI 4 / JSAPI
Audio
<<component>> Speech Recognition Engine
<<component>> Speech Recognition Engine

Figure 7: UML deployment diagram of application and framework

Application development is further simplified by a Struts-like MVC architecture. The application's forms are implemented as plug-ins, i.e., they implement a defined interface, follow a defined life cycle and are managed by a plug-in container. For a thorough discussion on component architecture and plug-in management refer to [Szy02].

As we mentioned before developing speech enabled applications needs special care in the sense of usability. Close interaction with the user is necessary to achieve a convient to use system. This inevitably leads to the fact that user interfaces and the business logic must often be changed. With the help of Gulliver it is possible to change the user interface of speech enabled applications as easy as of applications providing only a grahpical user interface. With the help of the custom multi-modal components that are integrated in a graphical designer costs and time for changes are minimized.

7 Conclusion and Further Work

Gulliver provides a framework that dramatically eases the development of speech-enabled applications. It allows application developers to enrich their user interfaces with speech and it neither causes any additional effort nor does it demand proprietary design concepts. Moreover, a developer does not need to take care of mobile device or speech recognition deployment specifics in the business logic layer or user interface. Nonetheless, the framework leaves room for improvement.

Currently, we implemented the framework in C++ (the audio handling and remote communication) and C# (the multimodal user interface components) targeted for Microsoft Windows Mobile Pocket PC and Smartphone Edition. For the implementation of the Gulliver Server we used Java. The various languages and platforms were chosen to show a proof of concepts.

As stated before, we based our implementation on scheme 2 as described in section 4. The audio data was compressed and transmitted to the remote server using VoIP. On the server side Nuance Dragon Naturally Speaking 9 was chosen as speech recognition engine. Besides recognition rate the required bandwidth is an important criterion. Not surprisingly the transmission of the audio data consumes most bandwidth (1.9 KB/s) and is responsible for 90% of data being transmitted. Due to the fact that VoIP uses Real Time Protocol (RTP), which itself is based on UDP, we can not guarantee delivery of complete audio data. Nonetheless even GPRS offers enough bandwidth to transfer enough audio data to achieve accurate recognition rates. Our testing scenarios showed that especially Dragon Naturally Speaking does not naturally reduce recognition rate but the speech recognition process takes more time. Using a high quality Bluetooth headset therefore leads nearly to the same recognition rate as using the system on the desktop with a headset.

However, as the Gulliver framework is designed not to be limited to a specific type of device and operating system, an application that uses the framework and which is targeted on a specific platform can potentially run on many types of devices and operating systems. The modifications to the application (e.g., switching the programming language) are laborious when they need to be done manually. An improved approach, which we are currently investigating, would in the first place describe an application on an abstract level (i.e. model) and then transform this abstract description (via intermediate steps) to different implementations. This is exactly the essence of model-driven software development as described in [BMS04]. We propose an abstract user interface model—platform independent model (PIM)—which defines the basic work flow of the application and its common properties. The abstract model could be transformed into more specific models called platform specific models (PSM) and finally into source code for a specific device and operating systems and necessary configuration files. An additional benefit from this approach is that many of the artifacts in our framework could be generated: for example, the plug-in configuration file or stubs for the application's forms could be generated from the application's work flow defined in the PIM (for further details please refer to [KMPS06]).

Acknowledgements

This research was supported by the Austrian Research Promotion Agency under the FH-plus program, the Austrian Broadcasting Corporation (ORF), and Microsoft. Any opinions, findings, and conclusions or recommendations in this paper are those of the authors and do not necessarily represent the views of the research sponsors.

References

[ACF+05] J. Axelsson, C. Cross, J. Ferrans, G. McCobb, T. V. Raman, and L. Wilson. Mobile X+V 1.2. http://www.voicexml.org/specs/multimodal/x+v/mobile/12/, 2005.

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture—A System of Patterns*. John Wiley & Sons, 1996.

[BMS04] S. Bleul, W. Mller, and R. Schaefer. Multimodal Dialog Description for Mobile Devices. In *Proceedings of International Conference on Advanced Visual Interfaces (AVI 2004)*, Gallipoly, Italy, 2004.

[Bro06] R. Brown. Talking Windows—Exploring New Speech Recognition And Synthesis APIs in Windows Vista. *MSDN Magazine*, 21(1), 2006.

[Fow03] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, Boston, 2003.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, 1995.

[HM04] A. Hunt and S. McGlashan. Speech Recognition Grammar Specification Version 1.0. http://www.w3.org/TR/speech-grammar/, 2004.

[KMPS06] W. Kurschl, S. Mitsch, R. Prokop, and J. Schönböck. Model-Driven Development of Speech-Enabled Applications. In *Proceedings FH Science Day*, pages 216–223, Hagenberg, Austria, 2006.

[KMPS07] W. Kurschl, S. Mitsch, R. Prokop, and J. Schönböck. Gulliver—A Framework for Building Smart Speech-Based Applications. In *Proceedings of the 40th Hawaii International Conference on System Sciences (HICSS-40)*, Hawaii, USA, 2007. IEEE.

[MBC+04] S. McGlashan, D. C. Burnett, J. Carter, P. Danielson, J. Ferrans, A. Hunt, B. Lucas, B. Porter, K. Rehor, and S. Tryphonas. Voice Extensible Markup Language (VoiceXML) Version 2.0, W3C Proposed Recommendation. http://www.w3.org/TR/voicexml20, 2004.

[Mic] Microsoft. Microsoft Speech SDK (SAPI 5.1). http://www.microsoft.com/speech/techinfo/apioverview/.

[MPS03] G. Mori, F. Paterno, and C. Santoro. Tool Support for Designing Nomadic Applications. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI)*, Miami, FL, USA, 2003. ACM Press.

[Pea00] D. Pearce. Enabling New Speech Driven Services for Mobile Devices: An Overview of the ETSI Standards Activities for Distributed Speech Recognition Front-ends. In *Proceedings of AVIOS 2000: The Speech Applications Conference*, San Jose, CA, USA, 2000.

[PS02] F. Paterno and C. Santoro. One Model, Many Interfaces. In *Proceedings of 4th International Conference on Computer-Aided Design of User Interfaces (CADUI)*, pages 143–154, Valenciennes, France, 2002. Kluwer Academics.

[Sun98a] Sun Microsystems, Inc. Grammar Format Specification. http://java.sun.com/products/java-media/speech/forDevelo%pers/JSGF/, 1998.

[Sun98b] Sun Microsystems, Inc. Java Speech API. http://java.sun.com/products/java-media/speech/, 1998.

[Szy02] C. Szyperski. *Component Software—Beyond Object-Oriented Programming*. Addison-Wesley, Boston, 2002.