Gulliver—A Framework for Building Smart Speech-Based Applications

Werner Kurschl
Upper Austria University of Applied Sciences
College of Information Technology
Hauptstr. 117, A–4232 Hagenberg, AUSTRIA
werner.kurschl@fh-hagenberg.at

Stefan Mitsch, Rene Prokop,
and Johannes Schönböck
Upper Austria University of Applied Sciences
Research Center Hagenberg
Hauptstr. 117, A–4232 Hagenberg, AUSTRIA
{*smitsch, rprokop, jschoenb*}*@fh-hagenberg.at*

Abstract

Speech recognition has matured over the past years to the point that companies can seriously consider its use. However, from a developer's perspective we observe that speech input is rarely used in mobile application development, not even if it allowed users to work with their devices more flexibly. This stems partly from the fact that programming speech-enabled applications is tedious, because there is insufficient framework and tool support.

This paper describes a component-based framework that uniformly supports development of multimodal applications on heterogeneous devices, ranging from laptop PCs to mobile phones. It especially focuses on distributed components (each performing a single step in speech recognition) to enable speech recognition on any type of device. Moreover, it describes how to develop and integrate different user interfaces for one application (voice-only, graphical-only, and multimodal) in a model-driven development approach, to minimize development and maintenance costs.

1. Introduction

An increasing number of businesses with mobile workers desire more flexibility and closer contact among their employees. For example, field workers that spend most of their time out of office and traveling between several sites need to communicate with in-house personnel; both groups utilize different devices: while stationary workers can rely on desktop PCs and fixed networks, field workers are constrained to mobile phones, personal digital assistants (PDAs), and limitations on laptop use. In order to better connect these two groups, the exchange of data between devices can still be improved and user input must be simplified. In this work we concentrate on applications for PDAs (PocketPC running Microsoft Windows Mobile 5.0), because we think they are the best compromise between features and size. In fact, recent technological developments in mobile devices (e.g. ubiquitous network access) have greatly advanced this so that both stationary and mobile workers use software as a substitute for direct contact with their coworkers; however, what they truly desire is to deal with the device in the same manner as they deal with coworkers, namely oral interaction.

The means to this end is speech recognition, which has advanced dramatically over the past years due to increased processing power and improved algorithms. Yet, it has not been adopted by a wide range of application domains, because keyboard and mouse are well-established input devices that are even emulated by computing devices lacking those input capabilities (e.g., PDAs). Based on the principles of Ubiquitous Computing, the device should move into the background to become an invisible helper that is operated along the way instead of being the central unit of work which demands the user's full attention. From that perspective the conventional input devices—like emulated or small-sized keyboards—are tedious to use, whereas speech input is the preferred means of input.

Of course there are also overwhelming reasons to use speech input. For example, it currently enables visually/physically handicapped people to gain access to software systems. It is also advantageous in situations where hand and eye distraction could be dangerous, e.g., driving. Moreover, Srinivasan and Brown have already documented in [20] that vertical markets such as customer service, travel, insurance, and banking have successfully integrated speech recognition systems into their enterprise applications.

Be that as it may, from a software development perspective programming speech-enabled applications is time-consuming due to the fact that there is insufficient framework and tool support (speech recognition engines and APIs only provide low-level events without semantics).

Speech recognition systems can be roughly categorized by the following criteria:

- Speaker-dependent or speaker-independent
- Constrained (grammar-based) or unconstrained (dictation-based) vocabulary

Today's speech recognition systems with an unconstrained vocabulary are speaker-dependent and demand high processing power and memory. Although the processing power of PDAs and mobile phones increases rapidly, it is far from being acceptable for unconstrained speech recognition. To date, no unconstrained speech recognition engine for such devices is available. Due to the complexity of recognizing unconstrained speech, most of these systems need to be trained to gain acceptable recognition rates of 95 percent and more. By contrast, constrained vocabulary speech recognition systems demand less powerful devices; they are often speaker-independent and available on PDAs as well.

The diversity of devices to be utilized in better connecting the groups mentioned above is a challenging factor in implementing software systems. Users expect the look-and-feel of a software system to be uniform across all platforms. Unfortunately, mobile devices are constrained in their processing power, memory, and input capabilities. Often applications created for a desktop PC are transformed to applications that are suitable for mobile devices. This transformation when done manually is laborious, error-prone, and costly. Thus there is a strong demand to create an application only once on an abstract level, and automatically transform this abstract model to various end devices.

This paper describes a component architecture that supports the development of multimodal applications on heterogeneous devices, ranging from laptop PCs to mobile phones. It's primary focus is on distributable components that enable fully featured speech recognition—constrained and unconstrained—on any kind of device. To accomplish the above-mentioned, we split the process of recognizing speech into small steps that can be distributed flexibly based on a device's capabilities. For example, laptops and tablet PCs are capable of performing all the steps in speech recognition, whereas a PDA can only perform constrained vocabulary speech recognition and needs to forward unconstrained speech recognition to a server. Our framework, called *Gulliver*, includes components for recording voice input from different sources (microphone or Bluetooth headset), for transporting voice data with VoIP over wireless networks, for server- and client-side speech recognition, and for a user interface with a unified event model for verbal and graphical user interaction.

Based on the model-driven software development principles a graphical editor provides the possibility to create an abstract user interface model that can be transformed to various end devices and different programming languages.

2. Related Work

Aurora Distributed Speech Recognition (DSR, see [17]) 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 (see [13]) 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 [5]), 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, see [12], must be supported by VoiceXML platforms).

Speech Application Language Tags (SALT, see [19]) is an alternative to VoiceXML. It is an extension of HTML and other markup languages that adds a speech interface to Web applications and services. SALT, like VoiceXML, also requires a special browser (either voice only or multimodal) that is capable of interpreting the SALT-specific tags; it does not support dictation, either.

XUL (see [10]) and UIML (see [4] and [18]) are both user interface markup languages based on XML. User interfaces are described in XML format and transformed into various user interface implementations by renderers, e.g., into a multimodal, VoiceXML-based user interface.

Microsoft Speech API (SAPI) 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, as stated above, there are no speech recognition engines that support dictation for these devices. And even if there were such engines, SAPI is not a user interface development environment (see [2]). It is located one layer below and solves the problem of unifying API access to different speech recognition engines.

Microsoft Windows Vista (see also [7]) 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 (see [16] and [14]) 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.

The MONA (see [15]) research group has developed a multimodal presentation server which basically generates a device-specific implementation of a user interface based on an abstract description. The format is a MONA-specific UIML format that is transported to the mobile device via Web services. Unfortunately the mobile device must be capable of interpreting this user interface description in a specific browser. If a multimodal user interface is needed, then VoiceXML is used as a target language.

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. Although there is a lot of research done in the automatic transformation of models to user interfaces all related work suggests using VoiceXML as language for multimodal 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.

Furthermore, our analysis showed that none of the examined voice user interface description languages (VoiceXML, SALT, etc.) support dictation; they are all restricted to constrained speech recognition, which limits their application domain. The current limitations of existing approaches prevent a seamless model and framework, which fits to different devices with different capabilities, so that speech-enabled application can be easily developed and be used better in the mobile application domain.

3. Speech Recognition on Mobile Devices

Developing speech-based applications on mobile devices requires a separation of different aspects, which we discuss in the following section.

3.1. Components of Speech Recognition

Speech recognition consists of three basic components, shown in Fig. 1, regardless of which device is used.

Digitizing: The spoken language is recorded by a microphone and transformed into a digital signal.

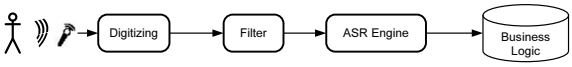

Figure 1. Speech recognition components.

Filtering: The digital signal is processed by audio filters, which, for example, minimize background noise, correct microphone typical distortion, or compress audio data.

Automatic Speech Recognition (ASR) Engine: The speech recognition engine turns the digital speech data into text.

These components are linked by transmission channels that send the audio data from one component to the next. Although this transmission might be a simple stream when the microphone and the speech engine are part of the same device, it could also be a VoIP or GSM channel when the speech recognition is placed on a remote machine.

3.2. Speech Recognition Schemes

To overcome the limitations listed in section 1, speech recognition components can be distributed across several machines.

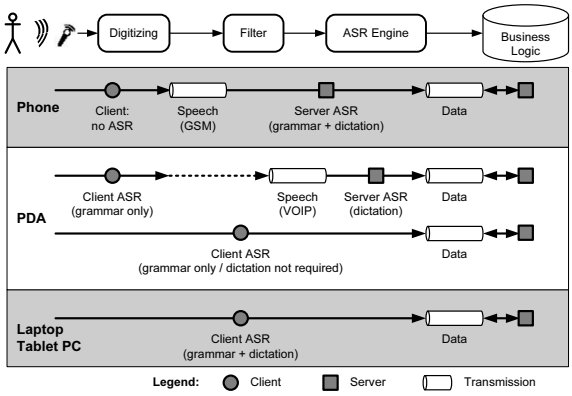

Figure 2. Speech recognition schemes.

Based on the components of speech recognition (described in section 3.1), three deployment schemes (depending on the type of mobile device) can be identified, as shown in Fig. 2:

Scheme 1—Standard or mobile phone: Full-featured voice recognition including dictation is available for neither mobile phones nor standard ones. Moreover, it is not possible to install additional software on standard phones. The only usable feature supported by all phone types is the transmission of spoken language using conventional line telephone networks or GSM. Thus, the phone's mic is the only speech recognition component used for digitizing the input. The filtering processes and the speech recognition engine must be placed on a remote machine. Additionally,

this scheme restricts the user to voice-only interfaces—the only communication channel available is the telephone network.

Scheme 2—PDA or Smartphone: These devices make possible the implementation of a simple graphical or even multimodal user interface and a grammar-based speech recognition engine. However, if dictation is a required feature of a speech-enabled application, an additional speech recognition engine must be placed on a remote server. As opposed to scheme 1, here the digitizer and filters can be placed on the mobile device, and the transmission protocol is not dependent on the telephone network any more—instead speech might be transmitted using alternative protocols like VoIP. As long as there is no need for dictation, all three components of speech recognition can be placed on the mobile device since the computation power is sufficient to do this locally.

Scheme 3—Laptop or Tablet PC: Portable personal computers, laptops or tablet PCs, provide enough computational power for local speech recognition, including dictation. Therefore, all three components of speech recognition can be deployed locally without any feature limitations.

3.3. A Speech Recognition Framework for Mobile Devices

A framework aimed at serving different types of client devices in an adequate way must be able to deal with each of these schemes at several layers.

At the *input processing layer* the translation of spoken language into text must be supported. This is represented by the basic speech recognition process described above. At the *user interface layer* the framework needs to support user interfaces of variable complexity. Powerful mobile devices may use speech just as an add-on to their conventional graphical user interfaces. When using standard phones, however, speech is the only way to interact.

The architecture described below defines how to implement these layers for each of the three schemes.

4. Architecture

As a result of the different requirements and schemes (see Fig. 2) of speech recognition on different types of devices, to obtain device-independent business logic and user interfaces, the input processing layer must be transparent. Fig. 3 reveals our architectural framework to accomplish this transparency.

Our architectural framework takes into account the fact that mobile applications are typically client/server systems: the business logic server usually is a separate server which is not part of the speech recognition architecture and is connected to the client's business logic only; by contrast, the speech recognition server does not implement any business logic. Of course these two servers might be deployed on the same remote machine.

In our architecture, the speech recognition process is hidden by the *Speech Input* component, which makes speech an input medium like the mouse or keyboard. The user interface and the business logic can use the Speech Input regardless of which features (e.g., dictation) are available locally and which must be deferred. The Speech Input component receives the audio data directly from a microphone or any other type of audio source and returns the recognized text or command.

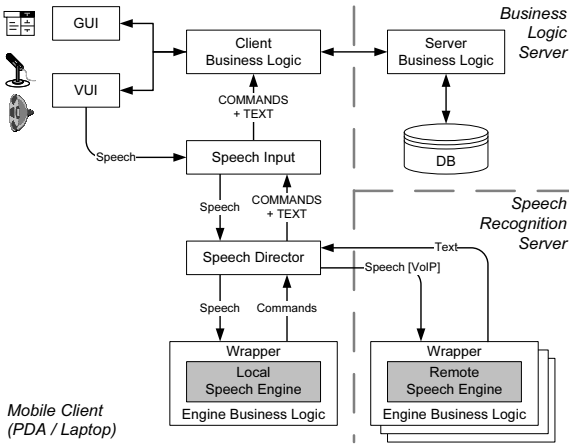

Figure 3. Speech recognition architecture.

The structure of the speech recognition process, which is hidden by the Speech Input component, depends on the processing power of the mobile device. Figure 3 shows the components that are needed for implementing speech recognition on PDAs, scheme 2. PDAs are able to perform simple command- or grammar-based speech recognition, but no dictation. To provide dictation as well, the dictation-based part of speech recognition is deferred to a remote speech server. As a result, the application gains access to fully featured speech recognition—regardless of the processing power of the mobile device—without any major disadvantages.

Of course the speech recognition process could be placed entirely on a remote server, as the DSR standard suggests (see [17]), but this would lead to an unacceptable effect: users expecting an immediate response in command-and-control would be confronted instead with an unacceptably long delay because of the time needed to transmit audio data over wireless networks.

By handling the time-critical part of speech recognition (i.e., the command and control features) locally on the mobile device, the response time is reduced to an acceptable range and the communication overhead with the speech recognition server is limited. Thus, the problem with la-

tency caused by a remote speech recognition engine occurs only when dictation is used. Nevertheless, users may accept this delay because of the time saved by not having to input data with a mouse, pen or keyboard, especially on a PDA.

The concurrent use of two speech engines requires a component that distributes audio data and merges the results. This is done by the *Speech Director*. It uses the local speech engine to recognize the user's commands. The switch to the remote speech recognition engine can be caused by verbal user commands, which are defined in a grammar and recognized by the local speech engine, by the client's business logic (e.g., when creating an e-mail), or by interaction with the graphical user interface in a multimodal application (e.g., setting the focus by mouse).

For the transmission of the audio data to the remote speech engine, VoIP is currently used. The audio data is passed from component to component (e.g., audio input to Speech Input, Speech Director to speech engine or VoIP transmitter) by *Speech Channels*, which are described in detail in section 5.1. These Speech Channels allow one a customizable linkage of components. Thus, single components—like the VoIP transmission—can be exchanged easily, or additional components—like resampling or noise reduction—can be inserted at any position.

The speech engines used on the mobile device and the remote speech recognition server are accessed through a *Speech Engine Wrapper*. This layer of abstraction is required when two or more speech engines (perhaps shipped by two different vendors) are used. A positive side effect of this is that the speech engines are exchangeable. Many speech recognition engines already support such a wrapper, e.g., Microsoft's Speech API.

The Speech Engine Wrapper receives audio data from a Speech Channel and provides it to the speech recognition engine. Moreover, it transforms grammars and commands that are delivered by the business logic in a generic format (e.g., SRGS) into the format supported by the speech recognition engine.

Apart from the Speech Input, most components are optional and depend on the type of mobile device used as well as on the required features. If there is only one local speech engine required (e.g., when using a laptop or when grammar-based speech recognition is sufficient), the Speech Director can be omitted and the Speech Input can be connected directly to the Speech Engine Wrapper.

5. Speech Processing Components

In the following sections we describe the key components of the Gulliver architecture in more detail.

5.1. Channels and Filters

The unpredictable target platform requires a highly flexible infrastructure for recording, transforming, and transmitting audio data. The recording feature has to support different audio sources, like a microphone built into a PDA or Bluetooth headsets. These different audio sources deliver audio data in different formats and quality. To suit the speech engine, some preprocessing steps (like re-sampling) must be performed. Depending on the type of mobile device, the preprocessing step is performed locally, otherwise it must be deferred to the remote speech recognition server.

To meet these requirements the transmission of audio data uses a pipes-and-filters (see [8]) architectural style. *Filters* are components that implement the speech processing functionality. The speech processing functionality ranges from simple tasks—like collecting audio data from an input device, normalizing the volume, or applying a codec to compress the audio data—to complex tasks like the transmission of audio data over wireless networks (e.g., using VoIP) and the speech recognition process itself.

A customizable linkage of these Filters must be possible to suit all schemes. Therefore, Filters are connected by point-to-point-pipes, called Speech Channels. The combination of Filters can be defined in a configuration and can easily be adapted to different types of devices.

Due to the good availability of speech processing implementations (like Bluetooth stacks as audio source, VoIP libraries for transmission, noise reduction algorithms, and speech engines), the implementation of a Filter is often reduced to developing a wrapper that can be connected to Speech Channels. We distinguish between two types of Filters regarding the way the data flow is handled: *(i)* those that provide data passively by waiting for the data to be collected by the successor—pull approach—*(ii)* and those that send data actively to the successor—push approach.

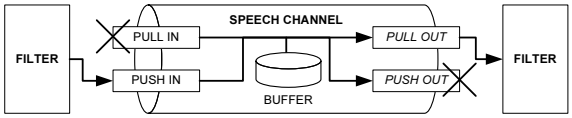

Figure 4. Self-adapting speech channel.

To enable a chain of different Filter types, a Speech Channel has to connect any type of Filters and adapt the data flow if necessary. For that requirement the Speech Channel is able to determine the operation type of the Filters and connect and adjust itself (see Fig. 4).

5.2. Speech, Grammar and Result/Event Routing

Almost all of today's GUI frameworks use an event model to communicate user input to an application's business logic. This paradigm is known by programmers and well established. To provide speech as a standard input device it has to be integrated in the same manner as the existing input devices so that the same concepts are followed and no differences are made. Thus, Gulliver provides multimodal (graphical and voice) user interface components that support speech while offering the same events as graphical user interface components.

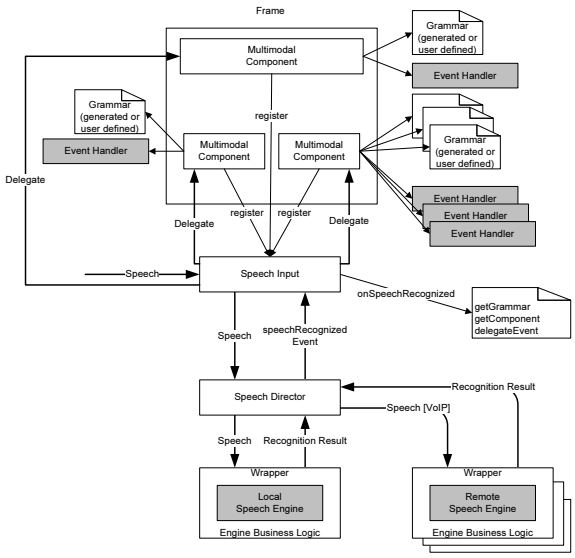

Figure 5. Speech input and event handling.

Currently, most speech recognition engines only provide a generic "speech recognized" event. Programmers have to deal with the details of the event; they often have to examine the recognized text to decide which action to take. This is very cumbersome and should be handled automatically by the framework. Within Gulliver the Speech Input component handles this specific problem as shown in Fig. 5. As soon as a multimodal component is added to the application it is registered at the Speech Input component. Handling a user interface with voice commands requires a grammar. In most cases this grammar can be generated automatically. A useful grammar for a button could, for example, be "Press the ok button" or simply "Press ok" if "ok" is the label of the button. This makes it possible to easily build and handle the user interface with voice commands from scratch. Additionally, programmers can provide custom grammars to define individual behavior. Typically, a user interface consists of a container called *Frame*, which groups several components. Every single component within such a Frame needs its own grammar. To ensure that every grammar can be identified and associated with its component, it automatically gets a unique identifier. When the Speech Input component receives a recognition result, it is thus possible to identify the component and the event the recognition result belongs to. The component then analyzes the result and invokes the corresponding standard event (e.g., the selection changed event if the component is a combo box). The business logic, which has to handle this event for mouse and keyboard input anyway, need not handle speech input separately—the speech input follows the same concept as used in GUI development.

A user interface typically consists of two types of components: those with a predefined set of valid input values (like combo boxes or date selectors), and those with unconstrained input (like text fields). Thus it must be possible to combine constrained (e.g., for combo boxes) and unconstrained speech recognition (for text fields) within a single application. The Speech Director mentioned above is responsible for routing speech data to the appropriate speech recognition engine. This component acts as a *Message Router* (see [11]).

The Speech Director can route speech data depending on one of the following paradigms:

- Focus (set by verbal commands, business logic, or interaction with a graphical user interface)
- Content-based message routing

Graphical user interfaces make use of the focus: for example, if you want to start typing text into a text field, you first have to set the focus to it (e.g., using the mouse). Instead of the mouse, voice commands could be used to set the focus. It is possible to define several keywords that refer to a graphical component. With the help of these keywords it is possible to create a grammar dynamically that allows the user to navigate through the user interface using voice commands. If a component supports dictation, the Speech Director has to switch to the remote speech recognition engine, which is able to do unconstrained speech recognition.

A more sophisticated approach would be a *Content-Based Message Router* (see [11]). It examines the message content and routes the message onto a different channel based on data contained in the message. But in this special case this would mean that the router itself has to be able to understand speech (which conflicts with the task of our router). A possibility is to define grammars that also contain free text. If the recognition result fits a grammar, the Message Router has to find out if the grammar does contain free text or not. If it does not contain free text, the recognition result can be passed to the Speech Input component immediately. Otherwise the spoken phrase and recognized text would have to be examined. For example, saying

the sentence "Write an e-mail to Person A with the subject Hello and the text Hello! I am sorry, I cannot attend the meeting today." would contain free text and grammar. The local speech recognition engine would inform the Message Router that this grammar contains free text. Therefore, the whole audio data would have to be sent to the remote engine, which is able to do unconstrained speech recognition, as well. Afterwards the recognized text of both engines would have to be divided into free text and commands needed to set the focus. This solution breaks some rules of the Content-Based Router pattern, because the router is not able to decide individually how to route the messages. It needs the help of some filters.

Therefore, we decided to route speech data depending on the focused user interface component. This not only is a well-known paradigm but it also guarantees shorter response times.

6. User Interface

In sections 4 and 5 we described the architecture of our framework. It's made up of flexible components that complete all the steps in speech recognition, and thus allows developers to concentrate on the core task at hand when creating a speech-enabled application. Furthermore, we introduced fully integrated multimodal user interface components that make it possible to build multimodal user interfaces in the same way as graphical user interfaces. Additionally, all these components can be implemented for various devices.

Supporting various devices, of course, in most cases leads to multiple user interfaces. For example, when using a standard telephone there is no graphical user interface, whereas on mobile phones there is at least a limited one, and on PDAs the graphical user interface is more sophisticated. Obviously, software developers must deal with the problem of constructing multiple versions of a single application, which are expensive in development and maintenance. Thus, a tool is needed that allows software developers to create different versions for different devices based on one common model. This is the essence of model-driven software development (see also [6]).

Our approach supports the generation of a concrete user interface, which is based on previously defined abstract models (see Fig. 6). It works in the following way.

The abstract UI model defines the application's control flow which is the same for all user interfaces. UML Activity diagrams (see [9]) provide appropriate modeling constructs, and XMI is the formal basis for transformations. Platform-specific models (PSM) are automatically generated through transformations; they can then be enriched with properties inherent to the platform. For voice-only user interfaces (without dictation), VoiceXML is an appropriate description for the PSM, whereas graphical-only user interfaces can be described through UIML, and multimodal user interfaces can be described using a combination of both. The PSMs may then be refined with additional models (e.g., laptop vs. PDA), and finally, source code is generated.

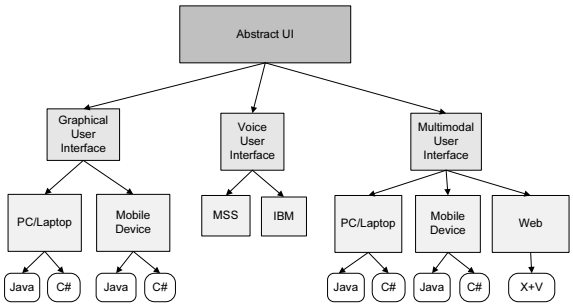

Figure 6. Transformation from abstract to concrete user interface models.

Multimodal user interfaces that do not support dictation (e.g., Web applications with voice support) can be implemented in X+V. But if rich-client applications with dictation support are needed, there is no implementation language available. Now, however, the proposed user interface components of Gulliver form the basis of such an implementation language.

To facilitate application development, we suggest a visual editor, comparable to a graphical user interface editor, to help the user visualize the final outcome. In the abstract model these elements can be rather simple (they just represent activities), but in the more detailed models the graphical representation should look similar to the real application on the device. An open source framework for creating a graphical editor is Graphical Model Framework (GMF, see [3]). With the help of this Eclipse framework it is possible to not only build feature-rich editors but also program the graphical representation of the model elements individually. Hence, it is quite simple to show different graphical elements in different models and to define a data model based on XML schema files.

UIML and VoiceXML are a perfect metamodel for our editor, because they already define how elements can be combined. The editor comprises several views—each representing one PSM—and automatically transforms these PSMs to more specific models, and finally to source code. To do this, we suggest using JET (Java Emitter Templates, see [1]) because it provides mature features.

7. Conclusion and Further Work

We have presented a component-based framework that supports the development of multimodal applications on

heterogeneous devices, ranging from laptop PCs to mobile phones. The framework's components split up the process of recognizing speech so that it can be flexibly distributed. Thus, fully featured speech recognition can be implemented on any device. The presented approach facilitates the development of speech-based applications for mobile application schemes.

In a further step we will enhance the user interface modeling tool to generate a more specific model or even source code for a specific platform.

8. Acknowledgment

We thank Gregory Curtis, Peter Schranz, Georg Schabetsberger, and Heiko Rahmel for their support and valuable input. This research was supported by the Austrian Research Promotion Agency under the FHplus 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

[1] JET Tutorial (Introduction to JET). http://www.eclipse.org/articles/Article-JET/jet˙tutorial1.html.

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

[3] The Eclipse Graphical Modeling Framework (GMF). http://www.eclipse.org/gmf/.

[4] M. Abrams. User Interface Markup Language (UIML) Draft Specification. http://www.uiml.org/specs/docs/uiml20-17jan00.pdf, 2000.

[5] 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.

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

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

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

[9] M. Fowler. *UML Distilled—A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, Boston, 2003.

[10] B. Goodger, I. Hickson, D. Hyatt, and C. Waterson. XML User Interface Language (XUL) 1.0. http://www.mozilla.org/projects/xul/, 2001.

[11] G. Hohpe and B. Woolf. *Enterprise Integration Patterns—Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, Boston, 2004.

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

[13] 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.

[14] 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.

[15] G. Niklfeld, H. Anegg, M. Pucher, R. Schatz, R. Simon, F. Wegscheider, A. Gassner, M. Jank, and G. Pospischil. Device Independent Mobile Multimodal User Interfaces with the MONA Multimodal Presentation Server. In *Proceedings of Eurescom Summit 2005*, Heidelberg, Germany, 2005.

[16] 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.

[17] 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.

[18] C. Phanouriou. *UIML: An Appliance-Independent XML User Interface Language*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, 2000.

[19] Saltforum. Speech Application Language Tags (SALT). http://www.saltforum.org/Saltforum/downloads/SALTTechnicalWhitePaper.pdf.

[20] S. Srinivasan and E. Brown. Is Speech Recognition Becoming Mainstream? *IEEE Computer*, 35(4):38–41, 2002.