An API for Voice User Interfaces in Pervasive Environments

Dirk Schnelle-Walka
Technische Universität Darmstadt
dirk@tk.informatik.tu-darmstadt.de

Stefan Radomski
Technische Universität Darmstadt
radomski@tk.informatik.tu-darmstadt.de

ABSTRACT

In this paper we present the Mundo Speech API, an application programming interface for voice user interfaces in pervasive environments. Distributed systems play a decisive role in pervasive environments but most existing implementations for voice based interaction still rely on the classical client/server paradigm or run on a single host. Building upon our publish/subscribe middleware, we present an extension for traditional discovery of devices in pervasive environments and a framework on top to implement voice user interfaces, taking into account the quality, changing availability and suitability of audio devices with regard to the user's context.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces And Presentation**]: User Interfaces – Voice I/O, Input devices and strategies; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – Query formulation; C.2.1 [**Computer - Communication Networks**]: Network Architecture and Design – Distributed networks

General Terms

Design, Human Factors, Performance

1. INTRODUCTION

Voice user interfaces (VUIs) are about to play an increasingly important role in pervasive environments. A user surrounded with innumerable devices may not be aware of each one and even less willing to use dedicated interaction panels for each. VUIs can alleviate this problem by providing an unintrusive way to interact with these devices through speech. While automatic speech recognition systems are improving, user acceptance remains unsatisfactory as word recognition rates with casual use are still to low to provide a convincing experience.

Many problems with speech recognition arise from the problem that the user's voice is too degraded by noise, reverb and other signal distortions when arriving at a speech recognition system, especially when users roam pervasive environments. In this paper we provide a set of concepts and a programming interface for voice user interface designers to automatically choose the most suitable sources and sinks for audio data with a roaming user. We present a – to our knowledge – novel approach to discover devices in the proximity of a user, to more reliably sample the users voice and improve upon automatic speech recognition as well as the understandability of the synthesized speech responses by using nearby speakers.

MobileHCI 2010 September 7-10, 2010, Lisboa, Portugal.
ACM 978-1-60558-835-3/10/09.

2. RELATED WORK

To implement user-tracking VUIs and to provide support for applications employing VUIs in pervasive environments, is to combine and extend existing solutions surrounding device discovery, user location and overall system architecture. There has been extensive work in each of these fields, but we feel that a comprehensive approach, combining all these solutions is still missing.

2.1 Universal Plug and Play

In 2000, Microsoft started the specification of Universal Plug and Play (UPnP), an architecture enabling connectivity for intelligent appliances. Today, the UPnP-Forum oversees the development and published version 1.1 of the specification in 2009 [17]. It has been of interest from its first appearance to facilitate plug and play speech technology [14] and did not loose its appeal today in the domain of smart homes [5]. Dimopulos et al. state that "UPnP technology [...] is currently the most established standard for device discovery and control in the home environment" [5].

In a UPnP environment, each device advertises its own description by broadcasting it to the environment once it is added to the network using a standard multicast address. Thus, control points listening to this address can discover these resources with their respective capabilities.

The UPnP device descriptions contain information of the devices and their services. For each service a description of commands, actions, data types and their ranges are specified. This information is used by the control point to generate SOAP messages to control the device.

Controlling a device also includes an understanding of the offered methods and capabilities of the devices. This is analogous to our approach, where the capabilities and services are discovered and maintained by a concept we call the *YellowPages* which is described later on. However, the control points still have to decide which of the offered services, if there are more of the same kind, is best suited to fulfill a certain task. This is not covered by UPnP. Imagine a user standing in his living room who wants to switch on a light by voice commands. Usually, there is more than one light to control and the control point has to clarify which of the

lights is concerned. This can be achieved by either initiating a clarification dialog or by asking a location service in the network and concluding that the relevant light is located in the living room. This location aware negotiation mechanism is built-in in our approach. Once the user issues the command *Turn the lights on!* the negotiation process starts and finds a suitable service that can fulfill this request in the proximity of the user. Hence, the API proposed in this paper offers support for recurring situations that are not covered by UPnP concepts but left to the specific implementation of the control points.

2.2 Mobile Speech Recognition

The limitations of mobile devices are still challenging for the support of voice based interaction with mobile devices. Several attempt have been made to work around them, resulting in a variety of architectures. Zaykovskiy analyzed them in [18], and categorized them into client, client-server and server-based architectures. However, in the service-oriented view of ubiquitous computing it makes more sense to emphasize the ability to have speech recognition as a network service or as an independent functionality of the device itself. This is a fundamental fact in pervasive environments, where services can become inaccessible while the user is on the move. This is also supported by Bailey [3] who requires that *"there need to be clear boundaries between the functionality of the device, and the functionality of the network"*.

Client based architectures are either implemented as hardware or as software solutions. Neither one offers the desired functionality for a successful deployment in pervasive environments. Hardware solutions are not flexible enough whereas speech engines implemented in software do not offer the performance of their desktop-based counterparts because they have to cope with limited computational power and available memory.

With server-based architectures, the audio recording capabilities of the embedded device serve as a microphone replacement to record the raw audio as the input for recognizer on a dedicated server. Usually, the audio is transferred as an RTP stream. Distributed speech recognition (DSR) is an architectural compromise of both worlds. DSR was introduced by ETSI [13], the European Telecommunication Standards Institute, wherein parts of the recognition process are performed on the device (DSR front-end) while the rest is handled on a server (DSR back-end) [13].

Compared to the pure audio streaming, the transmitted data is reduced without much loss of information. This also means that the error rates in transmission are reduced. As a positive consequence, DSR also works with lower signal strength which was shown in an experiment conducted in the Aurora project [12]. A great advantage of this technology over streaming solutions like RTP is the reduced network traffic.

DSR enables the capabilities of a desktop size recognizer on embedded devices, but it relies on a device that the user always carries with her. There are some situations where this is feasible, but in other situations (e.g. in a meeting) a headset is not applicable and the use of other architectures may make more sense. The Mundo Speech API enables the integration of all architectures using a single programming interface. Hence it is possible to choose the best architecture for the current situation on the fly.

Another architecture to enable speech in mobile and pervasive environments is Jaspis [16]. Jaspis is recognized as *"the most radical approach to architectures for spoken dialog systems"* [9]. It introduces an approach that *"reacts to changes in the information storage rather than on a model of turn-taking"*, enabling mixed-initiative dialogs. Although Jaspis employs an approach, wherein functional components can gradually be moved to the mobile devices, it still relies on a client/server paradigm. The centralized interaction manager coordinates not only the dialog flow but also input and output. The latter two are implemented as agents and are independent modules. This does not free them from knowing and discovering the available audio input and output devices that are available in the environment and which may change over time while the user is on the move. This is, where the Mundo API offers support. Hence, it could be used to extend the capabilities of the agents by supporting the discovery process of available audio devices.

Most of the existing systems decompose the functional components of voice user interfaces to a varying degree and deploy them using the traditional client/server paradigm. We hope that our approach of starting with a dedicated publish/subscribe middleware will ultimately result in a more scalable and robust framework for distributed voice user interfaces in pervasive environment.

3. VISION

The scenario, driving our requirements is a tracking voice user interface, using the most suitable means of communicating audio to and from a user depending on her location. In a prepared environment, there may exist several devices capable of sampling or rendering sound – on the move there often is a mobile device accompanying the user.

Imagine a user doing office work on a stationary computer system. While she is busy using her hands and eyes with traditional WIMP-based applications, she can use her voice as an additional interface to interact with the smart devices around her. At her desk, she wears a headset as she enjoys listening to music anyway. The attached microphone is unobtrusive and provides a decent quality for speech recognition and the voice memos she likes to record.

The calendar in her smart-phone realizes, that she has a meeting scheduled in 15 minutes and issues a reminder on her headphones. She acknowledges and asks the system to load the latest product presentation she has been working on onto her phone. She puts the headset away, as she feels awkward wearing it around her colleagues and leaves for the conference room. On her way, she realizes that she forgot the new logo design and uses the VUI in her smart-phone to retrieve them.

The conference room is prepared with beam-forming microphone arrays, tracking the users within using echo-location and IR enabled smart-badges. While the meeting takes place, the system automatically generates a protocol of the meeting for later reference.

To support such systems with employing voice user interfaces, our API provides concepts to query the surrounding of a user for nearby microphones and speakers. The ultimate goal is to have a distributed component, where systems can send text to be synthesized as speech and register their grammar to get notified when utterances, conforming the grammar are recognized. Thus allowing a wide range of devices with little regard to computational power to employ a voice user interface.

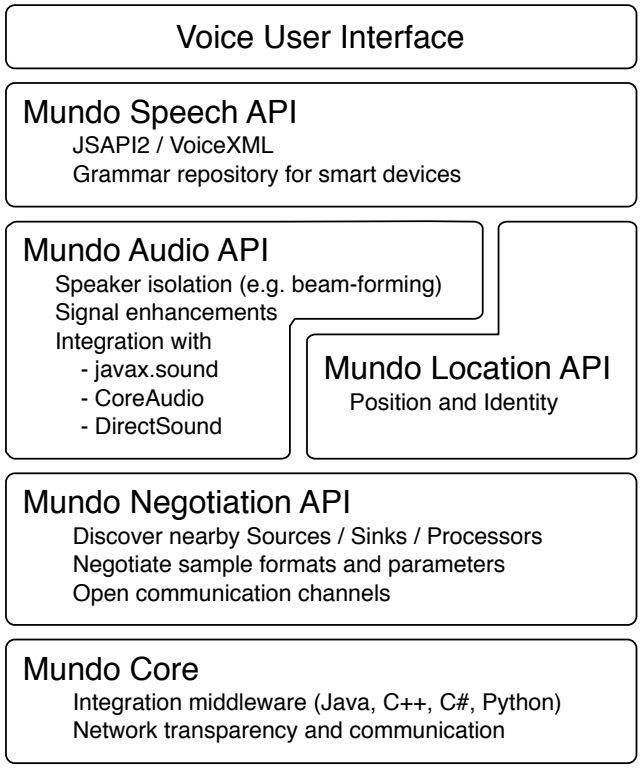

Figure 1: The layered model of the API

4. TECHNICAL BACKGROUND

The Mundo Speech API consists of several layers, implementing network transparency, discovery of functional components, negotiation of suitable parameters, allocation of resources and the actual speech synthesis and recognition (see Fig. 1).

The overall process to work with our API from the viewpoint of an application developer is as follows: First, a client gathers the functional components it intends to use or delegates the responsibility to find such components at runtime to special negotiators. All these components are then connected using the metaphor of contractual relations, initiated between the components by the requesting client. The discovery and composition of these components is done, using distributed remote procedure calls, while the actual transmission of messages is achieved by using channel-based publish/subscribe with a channel name, agreed upon per contract.

4.1 Network Transparency

The foundation for our API is formed by MundoCore [1], a light-weight middleware for distributed systems, available for a wide range of platforms from smart phones to dedicated servers. Building upon the publish/subscribe paradigm, it offers different flavors of network-transparent inter-process communication. Among them channel- and content-based publish/subscribe, synchronous and asynchronous remote procedure calls as well as service discovery. MundoCore is available in Java, C++, C# and has language bindings for Phyton and JavaScript, allowing for flexible deployments. Functional components in MundoCore are represented as services, implementing different interfaces, offering different functionality within a service landscape.

4.2 Queries and Discovery

On top of the network-layer, we offer the microphones and speakers within a pervasive environment as services available for discovery by requesting clients. Associated with each device are its supported audio formats as the valid combinations of the various audio related parameters (e.g. sampling rate, bit-depth, number of channels and valid encodings), its location, as well as a nested tree of arbitrary key-value pairs, describing additional information regarding the device. This information is managed by a set of entities called the *YellowPages*. They keep track of the functional components available and offer clients a way to query them. These entities can exist per participating node or may be delegated to nodes more capable (e.g. in terms of processing power).

4.2.1 Representing Capabilities and Location

In order to allow efficient queries for suitable audio sinks and sources in the proximity of a user, we developed a noteworthy approach for representing valid parameter combinations and locations. We conceive the set of relevant parameters to span a k-dimensional feature-space containing the valid parameter combinations represented as axis-aligned hyper-rectangles.

For example, a component to output audio on a pair of stationary speakers may support two different channels with a temporal resolution of 44kHz and a resolution for samples at 24Bit, but only a single channel with 8Bit if the temporal resolution is increased to 96kHz. To give an impression on the amount of valid parameter combinations with a single sink, the `javax.sound` implementation offers 72 different audio formats for the pair of speakers in a MacBook. While this is still a manageable amount, we hope to expand the approach to eventually allow queries for various resources within a smart environment.

Location is expressed as a three dimensional subspace, encoding the proximity of the device in a agreed-upon coordinate system. At the moment, we use WGS84 for the horizontal plane and a simple "meters above mean sea level" for height.

There has been extensive research regarding the storage and retrieval of k-dimensional point sets in a k-dimensional feature space, but less with regard to k-dimensional hyper-rectangles as objects with a non-empty volume. There are some generalizations of the various R-trees from 2D rectangles to higher dimensions [7] and some refinements of the index structures to support a large number of dimensions [4]. We decided, however, to settle for SKD-Trees, a generalization of KD-trees, supporting coverage and intersection [10].

While the runtime performance in the general case is worse as with the generalized R-trees, the implementation is trivial and easily validated. More elaborate data structures can be employed at a later date if the current representation proves to be a bottleneck for performance.

4.2.2 Querying for Components

A query for components is expressed as a hyper-rectangle with $\leq$ k dimensions, encoding valid ranges of values for the respective dimension and an optional filter object to further refine the search. Unset dimensions in the query rectangle are assumed to span an infinite range. The query rectangle

is intersected with the SKD-trees of all devices and those with an non-empty intersection are matched against the filter object. Components passing both tests are returned to the client as the matching results of a query.

4.2.3 Generality and Expressiveness

Finding a suitable compromise between expressiveness and runtime performance of filters and queries in publish/subscribe systems is an ongoing endeavor [6]. The most general approach is to allow arbitrary predicates, represented in a turing-complete formal language, evaluated against every message in such a system. On the other side of the spectrum are systems, organizing messages into topics or subjects, enabling clients to subscribe to or publish messages in these channels. Many more nuanced approaches exist between these two extremes, each with their respective advantages and disadvantages. We use the aforementioned queries only for discovery of devices and fall back onto channel-based publish/subscribe, once the parameters are negotiated and a channel name is created.

Conceiving the set of valid combinations of k different parameters as hyper-rectangles within a k-dimensional feature space poses some serious restrictions for the predicates available with queries. The most obvious one being, that they must be expressible as a set of intervals in $\Re^d$ (with $d \leq k$) at all. While this is the case for ranges of numeric values, such as the sample rate or the proximity of the device, it becomes less useful for enumerations such as available encodings or the endianness. One could introduce additional dimensions for every element within an enumeration, allowing only values of {0, 1}, but this gets cumbersome fast.

Our current approach is to pre-select a set of services as the intersection with the query hyper-rectangle and the optional filter object and then to refine the search on the client programmatically if needed, using the aforementioned tree of key-value pairs. With our approach, it is possible to formulate different, location-aware queries for devices such as:

- Find me a set of stereo speakers, capable of rendering 16Bit audio samples at 44kHz, no more than three meters away from this position.
- Get me all microphones in the conference room.

At the moment, only queries for actual audio devices are supported, but we hope to eventually offer queries for functional components in a more fine-granular decomposition of voice user interfaces and pervasive environments in general.

4.2.4 Providing Components

In order for components to be discoverable by the system, they need to be implemented as a service (in the Mundo sense), and provide callbacks to register their capabilities as the set of hyper-rectangles and their nested tree of key-value pairs. Both are registered by the framework to be available for queries at the *YellowPages*. Every component also implements `open` and `close`, which will be called and supplied with valid parameter combinations by the framework. We did not yet take into account concurrent use of components and it is up to every single component to handle subsequent calls to `open` with no `close` in between. Depending on the type of data processed, it may be possible to transcode and mix them into a single signal or simply indicate failure.

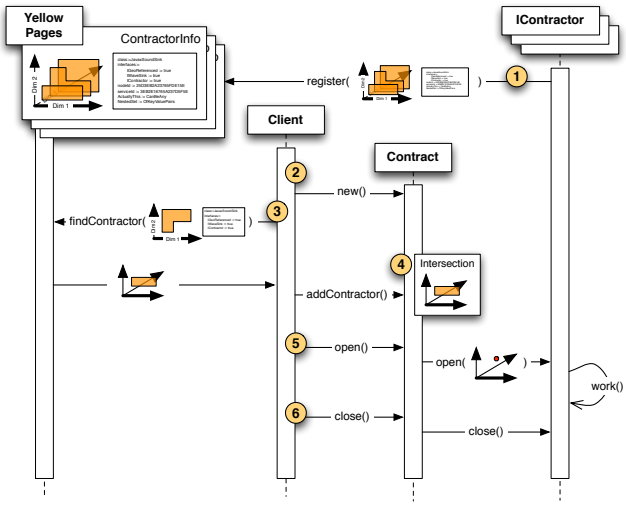

Figure 2: Process of contract negotiation

4.3 Negotiating Parameters

After a client gathered the set of microphones and speakers it intends to use, the audio format for the on-wire representation of samples has to be negotiated. While searching for suitable audio devices, a client has to ensure that there exists a common, valid parameter combinations for each participating device. We provide programmatic constructs and concepts for application developers to ensure this fact. Every single point within the intersection of the hyper-rectangles is a potential, valid parameter combination for every participating audio device. Again, we offer some convenience methods for application developers to select a suitable format or choose the default.

4.4 Contract Metaphor

The above mentioned concepts are refined into the metaphor of contractual relations, wherein *Contractors* advertise their capabilities as *ContractOffers* within a set of distributed *YellowPages*. Clients can request information regarding contractors by querying any instance of the YellowPages and add them into a *Contract*. The contract will only allow the addition of contractors whose capabilities have a non-empty intersection, ensuring an existing configuration for the various parameters every participating contractor is capable of (see Fig. 2).

A client can instantiate a contract and start adding contractors. Each contractor can provide presets as assignments of values to the different parameters along with a special default preset. When adding contractors, the contract will try to preserve the presets as long as all contractors can fulfill the given assignment of values. In addition to the various presets and the default, a client can chose to fix parameters within the contract, as long as the resulting assignment is included in the capabilities of all contractors. Fixing values by the client will inhibit the addition of unable contractors in the future. In any case, when all contractors are added and a subset of the values are set, there will be a valid assignment of values all contractors are capable of.

To allow for dynamic renegotiation of contractors during the lifetime of a contract, a client can choose to add *Negotiators* as well. These will continuously query for more

suitable contractors and add/remove them to the contract. For our demo application, the most important negotiator we implemented is the *ProximityNegotiator*. It is instantiated with a filter object and an optional query rectangle and searches for the closest contractor whose capabilities have a non-empty intersection with the potential valuations of the contract and the given query rectangle.

5. PROOF OF CONCEPT

We used the API to implement a prototypical application, wherein synthesized speech is rendered on speakers in the proximity of a roaming user. By using the Java implementation, we can already deploy components on a wide range of devices, among them a Viliv ultra-mobile PC (UMPC) and of course desktop PCs.

The setup consists of two desktop PCs and the UMPC with an IR reader as part of the IRIS system [2] and a headset, accompanying the user. For the demo, the client initiating the voice user interface is running on one of the desktop PCs and receives updates on recognized IR tags from the UMPC on a publish/subscribe channel from the MundoCore middleware. The application is modeled by offering the speakers of the three hosts as components for the above mentioned discovery and negotiation process. The client instantiates a single contract with a ProximityNegotiator, using the locations of the IR tags, as known a-priori, to set the subset of dimensions regarding the users proximity in the query rectangle. The subset with regard to the audio format consists of a single point as the audio format supported by the FreeTTS[1] speech synthesizer. Whenever a new IR tag is discovered, the proximity subset is updated and a query for nearby audio sinks is issued by the negotiator, resulting in the closing of the connection to the previous sink and the opening of a connection to the new one. In order to send audio data to the contracted devices, we implemented a Java OutputStream which can be instantiated with a negotiated contract[2].

We conducted no formal evaluation, but switching the output between the headset and the stationary speakers felt snappy and encourages us to test the approach with more elaborate scenarios in the future.

In addition to the Java implementation, we also developed an audio sink component based on the AudioQueues within the native CoreAudio framework of Macintosh computers, the iPhone and the upcoming iPad. While it is functional, using the iPhone as a mobile device proved to be problematic, as there is not yet an IR receiver and the CoreLocation framework is insufficient to determine the user's location with satisfactory accuracy. An IR receiver for the iPhone will become available within the next few month though[3].

6. PERSPECTIVE OF THE DEVELOPER

Current approaches, supporting speech in mobile and pervasive environments have only little focus on the authoring of such interfaces. We believe that such systems can only be successful if they are relatively easy to develop. Today, we find voice based applications mainly in telephony or

[1] `http://freetts.sourceforge.net`

[2] A video of our prototype can be viewed at `http://www.tk.informatik.tu-darmstadt.de/de/research/talk-touch-interaction/mundo-speech-api/`

[3] `http://l5remote.com/`

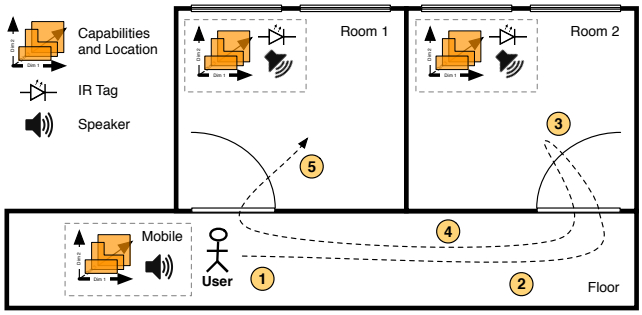

Figure 3: Demo Scenario

desktop environments. Both have different requirements for voice based interaction than pervasive environments. This makes it difficult for developers of such systems to reuse their knowledge in this new domain.

The following sections gives an insight on how developers for voice enabled desktop applications and telephony applications can reuse their knowledge with our API.

6.1 JSAPI 2

In 1998 SUN finalized the work on the first version of the Java Speech API (JSAPI) [15]. The API is supported in several open source speech engines like FreeTTS and Sphinx-4[4]. The missing concept of a streaming audio, inhibited it from being used in pervasive environments, where it is a basic requirement to transfer audio to and from the devices. JSAPI 2 [8] which was finalized as JSR 113 in 2009 removed this deficiency. It is designed to allow Java developers to easily incorporate speech technology in their applications. The JSAPI 2 primarily aims at the Java ME platform (specifically CLDC 1.0 and MIDP 1.0) and has a small memory footprint. Conversay states in the specification that *"implementations can require 0.5-1.5 MBytes of ROM for models and algorithms and approximately 128 KBytes of RAM depending on vocabulary and grammar size"* [8]. This makes it ideal to be used on small devices with limited computational capabilities. However, we use the Java platform only as a first reference implementation. The support of other programming languages like C++ and the languages from the .NET framework is planned as well.
The JSAPI 2 is event-based, which is an important requirement for mixed initiative dialogs in pervasive environments. It also fits nicely with the publish/subscribe paradigm of Mundo.

Since JSAPI 2 is a pure speech API it does not cover our need for a users location and context. Here, we make use of the *AudioManager* that is part of the JSAPI 2 to control sinks and sources by the means of a URL. The URL can contain the desired encoding to advise the underlying audio API to select a suitable port. We incorporate the scheme of the media locater that is defined in JSR-135[5] to notify the speech engine about the targeted audio format. Additionally, a stream object is passed to the audio manager that has been retrieved from the negotiation process. The speech engines make use of this stream object to send or receive the audio data. Our basic framework for the JSAPI

[4] `http://cmusphinx.sourceforge.net`

[5] `http://jcp.org/en/jsr/detail?id=135`

2 implementation is available as open source under the terms of the LGPL[6].

These concepts enable the development of voice based applications in pervasive environments while building upon the expertise of VUI developers that are familiar with this industry standard in desktop environments.

6.2 VoiceXML

Another well established standard is VoiceXML. The current version 2.1 [11] is designed for the development of telephony applications. VoiceXML is, due to its heritage for describing voice user interfaces with telephony systems, not geared towards the special requirements for dialog modeling in pervasive environments. Nevertheless, there have been several approaches to make use of it. One example is the INSPIRE system as it is described in [5]. The upcoming version 3.0 of VoiceXML[7] will also support the deployment apart from telephony, by using custom profiles. Profiles can contain a subset of modules that are suited for the target domain. VoiceXML 2.1 will be one of such legacy profiles. This direction of the W3C makes VoiceXML an interesting alternative as a domain specific language (DSL) for the development of future voice applications. Despite the new direction the W3C takes with this language, VoiceXML will still not offer sufficient support to meet the special requirements of voice based interfaces in pervasive environments.

JVoiceXML[8] is a voice browser that is able to interpret VoiceXML 2.1 documents. It features several implementation platforms to address the needs of the different speech engines. One platform is built upon JSAPI 2. Since this API is supported by default it is also possible to develop voice applications using VoiceXML with the Mundo Speech API.

We are implementing a solution based on these technologies in an ongoing project in an industrial setting.

7. FUTURE WORK

There are different areas to further improve upon our approach and the implementation. At the moment, we use filter objects against the trees of key/value pairs and query hyper-rectangles to match contractors within their feature spaces. It might prove to be beneficial to substitute the filter objects for already existing technologies such as OWL-S or UPnP and just extend their expressiveness.

Another area is to decompose the speech synthesis and recognition process itself and evaluate our approach for service discovery and composition of the resulting functional components. Additionally, components offering noise reduction and other signal enhancements will prove valuable and the question on how to compose more complex processing pipelines will arise.

Concurrent use of resources is another topic where we will have to conduct further research. At the moment, we simply ignore the issue and grant exclusive access to the various audio devices for the first client initiating the connection. In the future, we hope to improve our approach and develop a set of states and transitions for the contracts and contractors to support concurrent use and on-the-fly renegotiation of parameters when possible.

[6] `http://sourceforge.net/projects/jsapi/`
[7] `http://www.w3.org/TR/voicexml30/`
[8] `http://jvoicexml.sourceforge.net`

The next steps within our group are the implementation of beam-forming microphone arrays to conduct echo-location and speaker isolation, as well as the inclusion of additional devices to be used with VUIs. We hope to put away with the still rather clumsy UMPC in favor of modern smartphones running the Android or iPhoneOS platforms. Being able to track a user reliably and with sufficient accuracy will play a major role. At the moment, we rely on infrared tags, forcing a user to carry an infrared receiver for the system to recognize proximity to a tag. We hope to augment this approach by echo-location and speaker identification in the future.

In parallel to the framework supporting speech synthesis and recognition in pervasive environments, we will have to ponder about distributed dialog management. We hope to be able to use VoiceXML as a start and extend its concepts to support voice user interfaces spanning the various devices.

8. CONCLUSION

We presented our ongoing work regarding an application programming interface for voice user interfaces in pervasive environments. Building upon a representation of audio devices as a set of k-dimensional hyper-rectangles, encoding their supported audio formats and proximity, we presented an extension of service discovery in pervasive environments, extending the expressiveness of traditional technologies.

By using the newly introduced concept of audio streams in JSAPI 2, we were able to support part of a user tracking VUI with dialogs modeled in VoiceXML.

9. REFERENCES

[1] E. Aitenbichler, J. Kangasharju, and M. Mühlhäuser. MundoCore: A Light-weight Infrastructure for Pervasive Computing. *Pervasive and Mobile Computing*, 2007.

[2] E. Aitenbichler and M. Mühlhäuser. An IR Local Positioning System for Smart Items and Devices. In *Proceedings of the 23rd IEEE International Conference on Distributed Computing Systems Workshops (IWSAWC03)*, pages 334–339. IEEE Computer Society, May 2003.

[3] A. Bailey. Challenges and opportunities for interaction on mobile devices. Technical report, Canon Research Centre Europe Ltd., 2004.

[4] S. Berchtold, D. Keim, and H. Kriegel. The X-Tree: An Index Structure for High-Dimensional Data. *Proceedings of the 22nd VLDB Conference*, Jan 1996.

[5] T. Dimopulos, S. Albayrak, K. Engelbrecht, G. Lehmann, and S. Moller. Enhancing the Flexibility of a Multimodal Smart Home Environment. *Fortschritte der Akustik*, 33(2):639, 2007.

[6] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114ff., Jun 2003.

[7] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, Jan 1984.

[8] C. Hemphill and S. Rondel. JSR 113: Java Speech API 2.0. `http://jcp.org/en/jsr/detail?id=113`, May 2009.

[9] M. McTear. New directions in spoken dialogue technology for pervasive interfaces. In *Proceedings of the Workshop on Robust and Adaptive information Processing for Mobile Speech interfaces*, pages 57–64, 2004.

[10] B. Ooi. *Efficient Query Processing in Geographic Information Systems*. Springer Verlag, Jan 1990.

[11] M. Oshry, R. Auburn, P. Baggia, M. Bodell, D. Burke, D. C. Burnett, E. Candell, J. Carter, S. McGlashan, A. Lee, B. Porter, and K. Rehor. Voice Extensible Markup Language (VoiceXML) 2.1. `http://www.w3.org/TR/voicexml21/`, Jun 2007.

[12] D. Pearce. Enabling New Speech Driven Series for Mobile Devices: An overview of the ETSI standard activities for Distributed Speech Recognition Front-ends. Technical report, Motorola Labs, May 2000.

[13] D. Pearce. Enabling New Speech Driven Services for Mobile Devices: An overview of the ETSI standards activities for Distributed Speech Recognition Front-ends. In *AVIOS 2000: The Speech Applications Conference*, San Jose, CA, USA, May 2000.

[14] M. Rayner, I. Lewin, G. Gorrell, and J. Boye. Plug and play speech understanding. In *Proceedings of the Second SIGdial Workshop on Discourse and Dialogue*, pages 1–10, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

[15] SUN Microsysystems. Java Speech API Specification v.1.0. `http://java.sun.com/products/java-media/speech/reference/api/index.html`, 1998.

[16] M. Turunen, J. Hakulinen, K.-J. Räihä, E.-P. Salonen, A. Kainulainen, and P. Prusi. Jaspis an architecture and applications for speech-based accessibility systems. *IBM Systems Journal*, 44(3):485–504, 2005.

[17] UPnP Forum. UPnP Device Architecture 1.1, Oct 2008.

[18] D. Zaykovskiy. Survey of the Speech Recognition Techniques for Mobile Devices. In *11th International Conference on Speech and Computer (SPECOM)*, St. Petersburg (Russia), 2006.