Augmenting VoiceXML with Information from Pervasive Environments *

[Extended Abstract] †

Dirk Schnelle-Walka
TU Darmstadt - Telecooperation Group
Hochschulstr. 10
64289 Darmstadt, Germany
dirk@tk.informatik.tu-darmstadt.de

Stefan Radomski
TU Darmstadt - Telecooperation Group
Hochschulstr. 10
64289 Darmstadt, Germany
radomski@tk.informatik.tu-darmstadt.de

ABSTRACT

The expressiveness of VoiceXML to describe generic, spoken dialogs is, due to its heritage from telephony applications, rather limited in scope. There is no detailed information available regarding the environment of the dialog partners, hindering its applicability to model dialogs in pervasive environments. By introducing this information in the form of ECMA-Script variables, dialog authors can accommodate for the different scenarios and use-cases in pervasive environments. Thereby, extending the expressiveness of VoiceXML to describe more generic, spoken dialogs which take the users situation into account.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces And Presentation**]: User Interfaces—*Voice I/O, Input devices and strategies, Prototyping*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Distributed networks*

General Terms

Design, Human Factors

Keywords

Dynamic VoiceXML, ECMA Script, Voice User Interfaces

1. INTRODUCTION

Being part of the standardization efforts of the W3C for multimodal applications, VoiceXML is an established formal language to describe voice user interfaces. But with its roots in telephony applications, common requirements regarding an application in pervasive environments remain unsatisfied. While the `object` tag offers a syntactic element to manipulate potentially arbitrary devices in the environment, the way back, to capture information from the environment, is not directly supported. In [10] we already described our prototype to control devices in a home automation scenario with a dialog written in VoiceXML (see figure 1). Similar approaches were already around for a while but they still lack a descent way to utilize information from the environment. In this paper, we present our ongoing work of augmenting VoiceXML with ECMA variables to eventually support these requirements. Not unlike the transition from HTML/CGI to DHTML/AJAX for visual, dynamic web applications, we provide a more elaborate VoiceXML runtime by introducing suitable ECMA object models, containing information not about the VoiceXML document itself as with the DHTML DOM, but about the environment of the user.

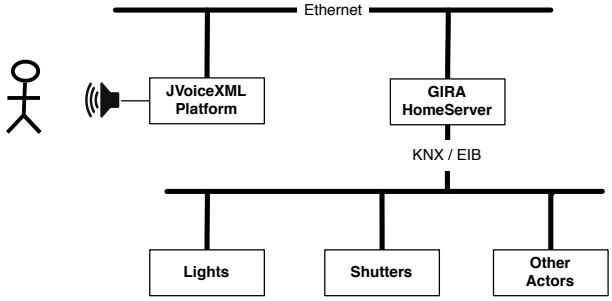

Figure 1: Controlling the environment in a smart home environment using VoiceXML (adapted from [10]).

2. RELATED WORK

Many frameworks to author voice based applications for pervasive environments do not make use of VoiceXML. One of them is the ODP (Ontology-based Dialog Platform) from SemVox, a spin-off of the German Research Center for Artificial Intelligence (DFKI GmbH). Multimodal input, like speech, touch or keyboard, coming from various clients is connected by semantic processing. ODP supports standards such as MRCP/SIP, EMMA and SSML and GUI frameworks

*(Produces the permission block, and copyright information). For use with SIG-ALTERNATE.CLS. Supported by ACM.

†A full version of this paper is available as *Author's Guide to Preparing ACM SIG Proceedings Using LaTeX2ε and BibTeX* at `www.acm.org/eaddress.htm`

MobileHCI Simpe '11 Stockholm, Sweden

such as Ajax, Flash/Flex or JavaFX as well as novel modalities (for example multitouch, gesture recognition or virtual characters) [1]. SemVox also provides its own workbench for rapid application development [12]. Applications are written in an proprietary XML format. Hence, existing design knowledge is hard to reuse. Although ODP introduces very expressive concepts they have to be learned and understood by the developers. The inherent complexities of ontologies are also often underestimated [3].

A system that makes use of VoiceXML is INSPIRE [4] from the Strategic Research Laboratories-Usabilitiy of the Deutsche Telekom Laboratories and the DAI-Labor, Berlin University of Technology. INSPIRE aims at a multimodal home automation system using e.g. voice and gestures. All interactions are controlled by the INSPIRE Core, leaving VoiceXML as a means to express the spoken presentation layer. The VoiceXML documents are generated from ontological descriptions. This also means that the knowledge about the environment is hard-coded into the generated documents and that they are valid only for the moment when they were generated up to the time when a timeout occurred to fetch the next document. Additionally, generated UIs are still imperfect and do not reach the quality of hand-crafted UIs [13, 14]. Thus, existing design knowledge is hard to use optimally for the generated UI case. Moreover, it is not possible to author mixed-initiative dialogs using this approach. User input is only accepted by simple fields that accept only a single piece of data, making it impossible to ask for missing data as it would be necessary in the following dialog taken from [10].

User: Please close the shutter
System: Which shutter shall be closed?
User does not know how to continue and says nothing
System: Do you want to close the shutter to the garden or to the terrace?
User: The terrace

A similar approach is made by OwlSpeak [5]. Like INSPIRE, VoiceXML documents are generated based on the knowledge that is captured in ontologies. OwlSpeak follows the information state update (ISU) model [6] reflecting the current dialog state as beliefs. A small VoiceXML document contains a field that refers to a grammar accepting all the possible commands to take the actions that are valid for the current belief. Since the way OwlSpeak uses VoiceXML is comparable to INSPIRE it inherits all the drawbacks regarding design quality and reaction to changes in the environment. In contrast to INSPIRE mixed-initiative dialogs are possible with OwlSpeak but rely on new concepts that come with ISU. Therefore, dialog designers have to learn the new concepts to transfer their existing design knowledge to the generation of dialogs using these concepts. It remains unclear if the gain of the ISU approach exceeds the loss in quality by a generated dialog.

3. APPROACH

For our approach, we extended JVoiceXML[2], with new ECMA variables reflecting an environment object model, containing various information about the current user and

[1] `http://www.semvox.de/en/menu-home.html`
[2] http://jvoicexml.sourceforge.net

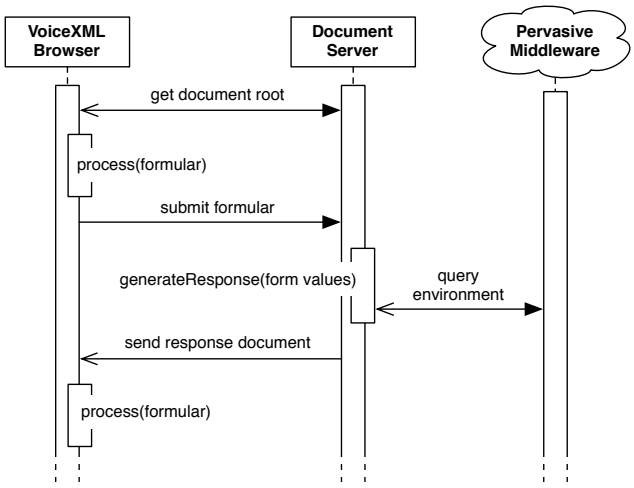

Figure 2: Dynamically generating a response document on the server after form submit.

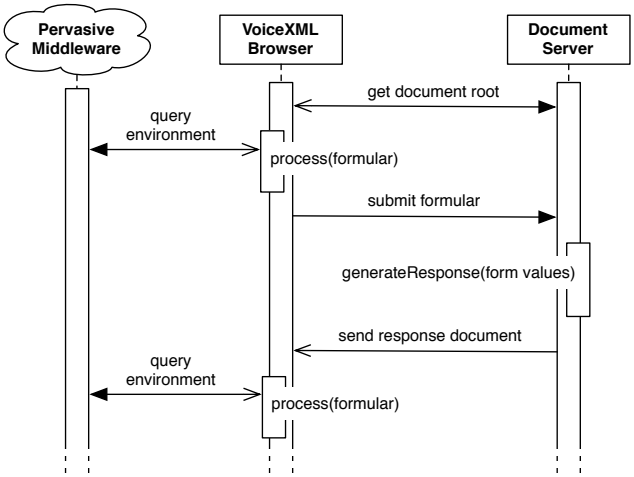

Figure 3: Using dynamic information from ECMA Variables.

her surroundings. There are different manners in which these variables can be used to direct the flow of a dialog:

- Conditional selection of a field element to be prompted by using such a variable in the fields `cond` expression.
- Selecting the next form by using such a variable in the `expr` attribute of a `goto` element.
- Pre-filling various fields depending on the state of the environment.

Since VoiceXML utilizes ECMA as the scripting environment throughout its elements, this list shows only some of the possible applications of the approach.

Pre-filling of fields will allow us to automatically assign values to those fields that can be collected from the user's context. This can be exploited, e.g. in mixed initiative dialogs. VoiceXML uses a slot-filling mechanism to enable mixed-initiative dialogs. When the dialog starts, the user

can provide all or no information for the slots to be filled. The VoiceXML interpreter will then ask for the missing data by visiting the fields that have not been assigned a value. They will also not be visited if they are filled by the context.

The overall approach is comparable to the transition from HTML/CGI (see figure 2) to DHTML/AJAX (see figure 3) for visual, dynamic web applications. Unlike AJAX'ified web pages where it is possible to change or replace the contents of XHTML DOM nodes, we deemed this unnecessary for VoiceXML documents, due to the one-dimensional nature of audio.

At the moment, our environmental object model does contain information about the user, and available audio in- and output devices (see listing 1).

```
"env" : {
  "user": {
    "location": {
      "x" = WGS84 Latitude,
      "y" = WGS84 Longitude,
      "z" = Meters above sea level
    }
  },
  ["source" | "sink"]: {
    "location": {
      "x" = WGS84 Latitude,
      "y" = WGS84 Longitude,
      "z" = Meters above sea level
    },
    "acceleration": {
      "x" = [-1..1],
      "y" = [-1..1],
      "z" = [-1..1]
    },
    "heading": [0..360],
  },
  "sink": {
    "audability": ["PRIVATE" | "PUBLIC"]
  },
  "sources": [all potential sources of audio],
  "sinks": [all potential sinks of audio]
}
```

Listing 1: Pseudo-JSON notation of the current environment object model.

The insertion of ECMA variables conforms to the scope mechanisms as it is defined by the VoiceXML specification [9] (see figure 4). The configuration for JVoiceXML is given in listing 3.1.

```
<variableprovider ...>
  <bean id="..." class="org.jvoicexml.
      interpreter.variables.VariableProviders">
    <bean class="org.jvoicexml.interpreter.
        variables.VariableProvider">
      <constructor-arg value="SESSION" />
      <entry key="env" value="org.jvoicexml.
          mundo.EnvironmentVarContainer"/>
    </bean>
    <bean class="org.jvoicexml.interpreter.
        variables.VariableProvider">
      <constructor-arg value="DOCUMENT" />
      <entry key="local" value="org.jvoicexml.
          mundo.LocalVarContainer"/>
    </bean>
  </bean>
</variableprovider>
```

Listing 2: Pseudo XML configuration for ECMA Variables.

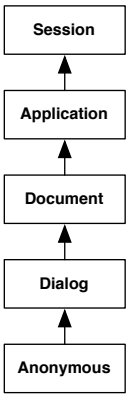

Figure 4: Scope hierarchy (adapted from [9])

Here, we defined two ECMA objects that are implemented as `ScriptableObject`s in Rhino[3] which is used as JVoiceXML's ECMA Script runtime. The corresponding ECMA script objects are instantiated when the given scope is entered. While the `EnvironmentVarContiner` is instantiated when the session is created, the `LocalVarContainer` is created if the document scope is entered. This allows for overwriting the default behavior per scope. For now, this configuration is JVoiceXML global, but we aim at having it configurable for each application, document etc.

Within the `EnvironmentVarContainer` we employ MundoCore [1] as our pervasive environment middleware, allowing us to connect to a wide range of sensors to retrieve the state of the user's situation. At the moment, the `location`, `heading` and `orientation` data is retrieved from CoreLocation framework as it can be found on current Macintosh and iOS platforms. Alternatively a plethora of other location tracking systems with implementations in MundoCore are available as well [2].

3.1 Short-Comings of Approach

- The VoiceXML specification does not account for asynchronous events. With the exception of barge-in, the platform itself can not throw arbitrary events to interrupt the processing of the current dialog step.
- The newly introduced ECMA objects are highly platform dependent and are not available with any platform but JVoiceXML.
- Invoking user-supplied callback functions in response to events from the pervasive middleware e.g.:

  ```
  env.sink.moved = function(acc) {...};
  ```

  does not work yet, as the ECMA variables in JVoiceXML are thread-local.

Especially the last point hinders us from reacting to context changes in an event-based manner as it would be needed for applications in pervasive environments. Imagine a user who picks up her phone to continue using the phone as the new input and output device instead of the previously used speakers and microphone of her desktop PC. While the employment of our Mundo Speech API (see figure 5) enables

[3]http://www.mozilla.org/rhino/

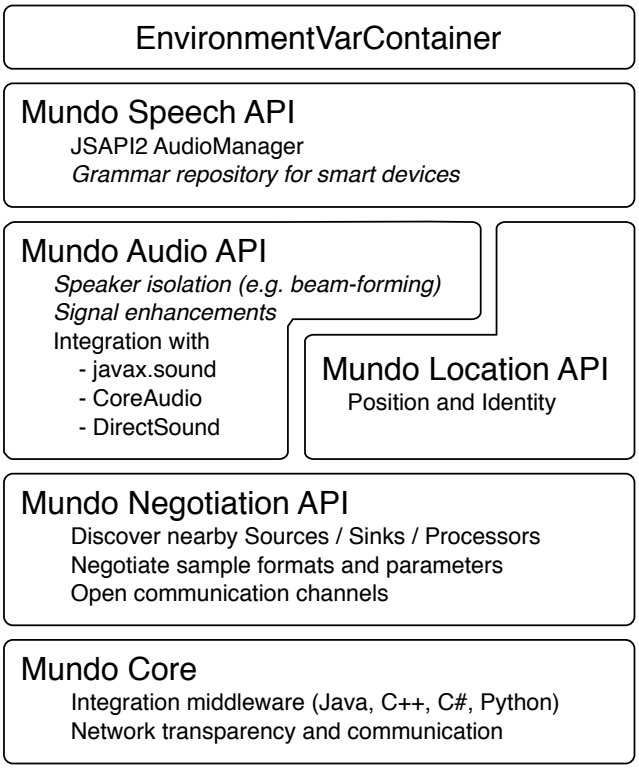

Figure 5: Layers of the Mundo Speech API [11]

us to provide such behavior nevertheless, it has to be implemented in the underlying layers of the variable containers and can not yet be expressed as ECMA script in the VoiceXML document.

This prevents us from e.g. explicitly asking for confirmation of this behavior. VoiceXML 2.x "...does not specify when events are thrown." [8]. While it is possible to have custom events within the VoiceXML that can be thrown programmatically via the `throw` tag, all events must occur internally making it impossible to react to external events. Furthermore, VoiceXML 2.x lacks a dedicated event life cycle, e.g. to run a specified script. This will change with the upcoming VoiceXML 3.0 [7]. VoiceXML 3.0 will feature an external communication module to send and receive external events; two new tags, `send` and `receive` are introduced which can be used asynchronously. The handling of asynchronous events is comparable to a `connection.disconnect` event that is thrown if the user hangs up her phone. It will also be possible to cancel the current media output, but not to modify it as it is possible with our approach where we can simply redirect the audio stream to another source or sink via a simple assignment of a value to an ECMA variable.

4. DEMO APPLICATION

As a proof of concept we developed a demo application. Imagine a group of friends discussing their next holiday trip. After some time they found an interesting spot and are about to book the flight and the hotel. They use the spoken dialog application built in the PC that they used to find interesting locations. The prompts are played over the speaker and the input is captured by a conference microphone that is placed on the table. After they conveyed all needed information the application announces the final price and one of them realizes that he has to check his account balance for the payment. Therefore, he continues with his telephone banking application using the same input and output devices. Since this information is private, the application queries the currently used device and detects that it does not have the audability set to `PRIVATE`. Therefore, the user is being asked to pick up his phone to continue the dialog on a more private channel. The handling of audio streams in this case is handled by our pervasive speech API [11].

5. CONCLUSION & OUTLOOK

Augmenting the JVoiceXML platform with ECMA object models, reflecting the state of a pervasive environment opens up a wide range of new applications, though our current implementation still leaves much to be desired. The inability to react to events in the environment within ECMA-Script diminishes the expressiveness as the corresponding behavior has to be modeled in the underlying layers; unaccessible to the dialog author. While it is possible to poll for changes (e.g. when a block element is entered), this is hardly a satisfiable solution which we hope to resolve in the future.

By introducing the environments state as ECMA object models, all concepts of VoiceXML remain intact, allowing dialog authors to add "pervasive behavior" as an after-thought to existing dialogs.

In the long term, VoiceXML 3.0 will be able to communicate with the environment using the external communication module. However, this is still something that has not been finalized in the current working draft. For now, we created an implementation that lets us experiment with a current VoiceXML browser implementation to interact with the environment using speech. The effects of this asynchronous methods introduced in VoiceXML 3.0 is still unknown. The authors of the specification are concerned that "...they can be disruptive to a voice application" [7]. This will not be the case for our implementation since we smoothly integrate into the scripting concepts of VoiceXML 2.x.

One of the next steps will be to integrate our approach into a home automation application that was described above. Here, our approach can also be used as a feedback channel to query the devices' states.

6. REFERENCES

[1] E. Aitenbichler, J. Kangasharju, and M. Mühlhäuser. MundoCore: A Light-weight Infrastructure for Pervasive Computing. *Pervasive and Mobile Computing*, 2007. doi:10.1016/j.pmcj.2007.04.002 (332–361).

[2] E. Aitenbichler, F. Lyardet, A. Hadjakos, and M. Mühlhäuser. Fine-grained evaluation of local positioning systems for specific target applications. In *Proceedings of the 6th International Conference on Ubiquitous Intelligence and Computing*, UIC '09, pages 236–250, Berlin, Heidelberg, 2009. Springer-Verlag.

[3] T. Albertsen and E. Blomqvist. Describing ontology applications. In *ESWC '07: Proceedings of the 4th European conference on The Semantic Web*, pages 549–563, Berlin, Heidelberg, 2007. Springer-Verlag.

[4] 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.

[5] T. Heinroth, D. Denich, and A. Schmitt. OwlSpeak - adaptive spoken dialogue within intelligent environments. In *8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 666 – 671, Mar. 2010.

[6] S. Larsson and D. Traum. Information state and dialogue management in the trindi dialogue move engine toolkit. *Natural language engineering*, 6(3&4):323–340, 2000.

[7] S. McGlashan, D. C. Burnett, R. Akolkar, R. Auburn, P. Baggia, J. Barnett, M. Bodell, J. Carter, M. Oshry, K. Rehor, X. Yang, M. Young, and R. Hosn. Voice Extensible Markup Language (VoiceXML) Version 3.0, W3C Working Draft. `http://www.w3.org/TR/voicexml30/`, Dec. 2010.

[8] S. McGlashan, D. C. Burnett, J. Carter, P. Danielsen, J. Ferrans, A. Hunt, B. Lucas, B. Porter, K. Rehor, and S. Tryphonas. Voice Extensible Markup Language (VoiceXML) version 2.0. `http://www.w3.org/TR/voicexml20/`, Mar. 2004.

[9] 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) Version 2.1, W3C Recommendation. `http://www.w3.org/TR/voicexml21/`, June 2007.

[10] D. Schnelle-Walka, J. Arndt, and S. Feldes. Towards mixed-initiave concepts in smart environments. In *Proceedings of Workshop Interacting with Smart Objects*, Feb 2011.

[11] D. Schnelle-Walka and S. Radomski. An api for voice user interfaces in pervasive environments. In *SimPE 2010 Conference Proceedings*, Sep 2010.

[12] D. Sonntag, G. Sonnenberg, R. Nesselrath, and G. Herzog. Supporting a rapid dialogue engineering process. In *Proceedings of the First International Workshop On Spoken Dialogue Systems Technology (IWSDS)*, 2009.

[13] N. Sukaviriya, S. Kovacevic, J. Foley, B. Myers, D. Olsen Jr, and M. Schneider-Hufschmidt. Model-based user interfaces: what are they and why should we care? In *Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 133–135. ACM, 1994.

[14] P. Szekely. Retrospective and challenges for model-based interface development. In *Design, Specification and Verification of Interactive Systems*, volume 96, pages 1–27. Citeseer, 1996.