Implementation of Dialog Applications in an Open-Source VoiceXML Platform

Ricardo de Córdoba, Fernando Fernández, Valentín Sama, Luis F. D'Haro, Rubén San-Segundo, Juan M. Montero

Speech Technology Group. Dept. of Electronic Engineering. Universidad Politécnica de Madrid
E.T.S.I. Telecomunicación. Ciudad Universitaria s/n, 28040 Madrid, Spain
{cordoba, efhes, vsama, lfdharo, lapiz, juancho}@die.upm.es
http://www-gth.die.upm.es

ABSTRACT

In this paper, we study the approach followed to use the VoiceXML standard in a dialog system platform already available in our group. As VoiceXML interpreter we have chosen OpenVXI, an open source portable solution where we can make the modifications needed to adapt the solution to the characteristics of our recognition and synthesis modules; so we will emphasize the changes that we have had to make in such interpreter. Besides, we review some relevant modules in our platform and their capabilities, highlighting the use of standards in them, as SSML for the text-to-speech system and JSGF for the specification of grammars for recognition. Finally, we discuss several ideas regarding the limitations detected in VoiceXML.

Keywords: automatic dialog systems, speech recognition, voice synthesis, VoiceXML, OpenVXI.

1. Introduction

The Gemini project[1] (Generic Environment for Multilingual Interactive Natural Interfaces), is a two year project that is just finishing with the following partners: Knowledge S.A. (Greece) as coordinator, Patras University – WCL (Greece), TEMIC SDS (Germany), UPM – GTH (Spain), FAW (Germany) and Egnatia Bank (Greece). It exploits the results obtained in the IDAS project ([1][2]) and from real-world use of similar systems, to create a generic platform for the development of user-friendly, natural, high quality, intuitive, platform independent and multi-modal interactive interfaces to a wide area of databases employed by information service providers.

The main objective of the project has been the development of a platform able to generate dialog applications in a semi-automatic way that are able to handle several languages and several modalities at the same time [3]. The second basic objective is the preparation of a real-time platform where the scripts generated in the design platform could be executed. These scripts for a speech application are written in VoiceXML 2.0.

To execute the scripts, we need an interpreter. We have chosen OpenVXI 2.0.1, from SpeechWorks [4], as it is an open-source solution and offers portability, as any speech recognizer or text-to-speech system can be used. There is no restriction for the telephone platform either. It has some drawbacks though: the documentation is rather reduced and there is not any functional implementation that can be used as a model for development.

The use of VoiceXML gives our system a high degree of standardization, thanks to the growing interest in VoiceXML as standard language for the design of speech applications.

To demonstrate the efficiency of the platform, it has been tested in a banking application with several services. It offers general information of bank products: personal loans, car loans, mortgages, deposits, credit cards, etc.; user authentication based in account number and PIN; account balance and last transactions information; and the possibility to perform transactions.

2. OpenVXI interpreter

The OpenVXI interpreter is an open-source software developed by SpeechWorks [4] that is used to execute VoiceXML scripts. Its main advantage is that it provides a big part of the functionality needed to execute dialog applications, as a XML interface that processes the VoiceXML script, a JavaScript API, an API that handles WWW operations and an API for registrations. For all functions related to input/output (recognition, synthesis and telephone control) it provides incomplete interfaces that can be modified to adapt them to the needs of each platform. Let us describe the work done in each module.

2.1 Speech recognition module

A great effort has been made to adapt our recognition software to OpenVXI. The system allows the dynamic loading and using of different recognizers, even simultaneously, during the execution of the VoiceXML application. We have implemented VoiceXML based mechanisms that identify the recognizer that has to be used in each prompt of the application. We can also activate different grammars in parallel. Using this functionality, we can use state-specific language models and acoustic models, so that we can adapt to different circumstances: continuous speech, isolated speech, connected digits, dates, etc.

To do that, following the directives of the VoiceXML 2.0 specification, we have defined a new property that will identify the recognizer to be used. This is an example:

<property name= "es.upm.gth.recognizer" value="Dates" />

The platform will load and use the recognizer specified (specific for Dates, in this case).

Our implementation of the recognition interface for OpenVXI has the following characteristics:

- Management and registration of errors and events (e.g. timeouts).

[1] This work was partly supported by the European Commission's Information Society Technologies Programme under contract no. IST-2001-32343. The authors are solely responsible for the contents of this publication.

Refer to the GEMINI Project Homepage on www.gemini-project.org for further details.

We have implemented the mechanisms needed to handle events as 'user asks for help', which can be related to the prompt provided by the system or with the state of the dialog, 'user wants to exit', 'user wants the prompt to be repeated' or 'system timeout after no speech detected in the recognition'.

- Several input channels

The module is designed so that it can use as input both the telephone line, a microphone or previously saved files. Our engine allows the wav files recorded during the recognition to be saved, so that they can be used afterwards for debugging purposes.

We will now describe the functionality that is not provided in OpenVXI and that we have implemented:

1. A channel identifier

We need an identifier for the channel currently used, so that we can assign exclusively a specific input audio device. We have assigned a number to each channel thread; so, we have needed to change the function headers to include that value.

2. Pre-loading of the recognizer

In order to save time in the real-time system, we need to load the recognizer data (models, structures, grammars, etc.) when the system is first run. There is no mention in VoiceXML or in OpenVXI regarding the loading of the recognizer, and there is no specific time or place where that loading can be done. We have identified two possible places to do it: during the loading of the grammar or in the first call to the recognizer. As the loading of the grammar is previous to the recognition, it is the place where we have finally decided to load the recognizer.

3. Grammar-recognizer link

We need a direct link between the active grammar(s) and the recognition models. In OpenVXI you can have multiple active grammars, as well as different recognition models, but the link between them is left open both in VoiceXML and OpenVXI. To let the interpreter know which specific models and grammars should be used in the recognition, we decided to use the implementation of the method *LoadGrammarFromUri* from the recognition interface VXIrec (empty in the original distribution of OpenVXI). This method is used to load a specific grammar when the attribute "src" in the label <grammar> is not null. This attribute indicates the URI direction of the document with the grammar definition. Our implementation includes the search and loading of that document. Moreover, we allow that a link between a grammar and some specific models be made using the attribute "type" associated to the label <grammar> used to declare the grammar.

<grammar src="data/bigram.dat" type="Isolated"/>

this line indicates to the interpreter the recognition models that should be loaded (referenced by "Isolated") and the grammar linked to those models ("data/bigram.dat"). Besides, any link between a grammar and some models can be established using a different "type" for each model.

2.2 Speech generation module

This engine offers all the services related to voice generation considered in the VoiceXML standard, including the reproduction of audio files and text-to-speech conversion to handle variable content messages. The voice generation engine consists of the following modules:

- Speech synthesis module. Created using the functions offered by "Boris", the text-to-speech system developed by GTH [5].
- Recorded speech player module. It allows the reproduction of speech files specifying the full path of the file.

The implementation also covers the following issues:

- The user is able to modify the prosody of the speech, using the tags specified in the SSML standard. To cope with this, several new functions have been developed to interpret SSML tags.
- It uses speech synthesis when the audio file specified is not available.
- The process is asynchronous, non-blocking, so that the user can interrupt the dialog (barge-in) according to the specification VoiceXML 2.0.
- Management and registration of the different errors and events.

The functionality that is missing in OpenVXI and that we have had to implement is the following:

1. A channel identifier

We also need a channel identifier in a similar way than for the recognition (see Section 2.1).

2. FIFO queue

We have had to implement a structure in the *Queue* procedure where to keep the voice messages that will be played when calling the *Play* procedure.

3. *Play* procedure

The official documentation of this procedure is incomplete for the non-blocking reproduction. What we need to do is: check if there is any message to be played (if not, exit); check if there is currently another playing process running (if yes, wait until it is over); and a loop where all pending messages in the queue are played in a blocking fashion except the last one that will be played asynchronously.

The main objective of the SSML (Speech Synthesis Markup Language) standard [6] is to provide a standard way to control the different aspects of speech synthesis, as the f0, the pronunciation, the volume, etc. It uses a set of labels that are inserted in the text to be synthesized. We have implemented the following labels in our synthesizer:

1. "emphasis" (to emphasize specific fragments), with the following values for the "level" attribute: "strong", "moderate", "none" and "reduced".
2. "break" (a break of a specific duration), with the "time" attribute, which is an integer followed by "ms", e.g., "300ms" to make a break of 300 ms.
3. "prosody", with "pitch", "rate" and "volume" attributes. For all of them, the values can be specified in three ways: (a) using discrete values with strings, like "x-high" || "high"|| "medium"||..., (b) with a floating point value and the corresponding unit, like "100Hz", or (c) using a relative value as a positive or negative percentage, e.g. "+10%". We have chosen this last option for its flexibility and portability to new voices.

Example:
¿Do you want to transfer <break time="90ms"/> <emphasis level="strong"> 100 euros </emphasis> from your account <prosody volume="+20%"> 34656 </prosody> to the account <prosody pitch="+40%"> 56454 </prosody>?

2.3 Telephone control module

The VoiceXML standard only specifies the actions to disconnect and transfer the calls, so the integration of the telephone services in our platform has been relatively easy.

The main function of this module is to connect the system to the telephone line and adapt the signal levels at both sides of the interface. It performs many other functions:

- Signal adaptation to connect input and output external devices, as loudspeakers, microphones or headphones.
- Control of connection/disconnection.
- Control of an external recorder.
- Call / Put down detection.
- DTMF detection.

We have implemented the corresponding control mechanisms for these functions in the OpenVXI interpreter, always following the specifications of VoiceXML 2.0.

We have also introduced a new and useful functionality: the simulation of the telephone line using a microphone and a loudspeaker. This way, we can test the system or make demonstrations using those devices instead of the telephone line with a simple change in a configuration file.

3. The platform modules

We will briefly describe some relevant modules available in our group to create a runtime platform able to execute dialog applications in real time.

3.1 Speech recognition module

Right now, our system uses continuous HMM trained with the SpeechDat database, with 4.000 speakers and some 46 hours of continuous speech. We use triphone models clustered using a decision-tree algorithm. The system has 1807 different states, with 6 Gaussians in each state. The error rate for a continuous speech telephone task using 3,065 words, is 4.2%.

Besides, in our system we use an isolated word recognizer with very similar characteristics to the previous one, specialized to isolated utterances. We also have recognizers adapted to the recognition of digits and spelling (we can use spelling when there are recognition errors).

We also have a module dedicated to compute confidence level values of the recognition, as it is absolutely needed to decide the confirmation type and the user level. If the confidence level is low, explicit confirmation should be used; if it is high, we could use implicit or no confirmation. This module uses parameters at word and sentence level offering high reliability.

3.2 Understanding module

The understanding module extracts the concepts that will be used by the OpenVXI interpreter to fill the structure that contains the recognition results.

We have adapted this module to the banking application developed in the project. The understanding is based in context dependent rules, using a single dictionary for all the prompts of the system (16 in total). This dictionary is semantically labeled based on the information that has to be extracted. Words that do not provide useful info are assigned the "garbage" category, just like the out-of-vocabulary words. Nevertheless, they can be used in the rules if needed.

To label the dictionary, we have taken into account that the same word may appear in different prompts. In addition, a word may have multiple categories, one of them "garbage", so that, if it is needed, the word can be transformed by the understanding rules before the elimination of "garbage" items. We think that this does not cause problems, as the understanding rules change in each prompt (prompt-specific rules) and the system knows the current state in the dialog, so that it only activates the correct prompt rules.

The sequence of actions done in this module can be summarized as follows: 1) input string pre-processing, including the assignation of categories, digit processing, interjections suppression, etc.; 2) rules applied before the suppression of words with category "garbage"; 3) suppression of "garbage" words; 4) rules applied after the suppression, 5) writing of the understanding concepts. Let us see an example:

1) Pre-processing.

2) rewrite3 ("ID_change", "garbage", "ID_currency", "currency_exchange");

3) Suppression of "garbage" words

4) rewrite2 ("ID_want", "currency_exchange", "SLOT_currency_exchange");

5) Writing the results.

We use rewriting rules that specify first the origin categories and, in the last position, the destination category. In the rule from step 2, three words specified in this order (the first ones) are going to be converted in an item of category "currency_exchange". This rule will gather expressions like "cambio de moneda" in Spanish. After the suppression of "garbage" words, the rule in step 4 converts the items "ID_want" and "currency_exchange" into the result "SLOT_currency_exchange" that is used in sep 5.

3.3 Language modeling module

We can use the usual stochastic language models. Besides, in this project we have developed JSGF (Java Speech Grammar Format) grammars. These grammars are platform-independent and are based in the concept of rule grammar adapting some Java conventions to the traditional specification of grammars.

As in the understanding module, we have a different JSGF grammar for each prompt of the system, besides one specific grammar for digits. As our system is multilingual, there are different grammars for each language available.

This is an example of JSGF grammar for a prompt in Spanish:

```
#JSGF V1.0 ISO8859-1;
grammar GeneralInformationCathegory;
public <infocathegory> =
    <Deposit_Products> {this.$value="DepositProducts"}|
    <Cards>          {this.$value = "Cards"} |
    <Exchange_Rates>         {this.$value="ExchangeRates"};
```

<Deposit_Products>= [infórmeme (sobre | de)] depósitos [<Polite>];
<Cards>= [<I_want>] tarjetas ;
<Exchange_Rates>= [<I_want>] cambio [<Currencies>] ;
<Currencies> = de (moneda | divisas)
<I_want> = [(quiero | deseo)] información (de | sobre);
<Polite> = (por favor);

This is the grammar for a prompt where the user selects between several banking products (deposits, credit cards or currency exchange information). The optional items are shown in square brackets and parentheses have been used to group or disambiguate expressions. Also, inside brackets we have labels with specific information of the application. Alternatives are expressed using '|'. For example, in the rule <Exchange_Rates> the item <I_want> is optional, so they may appear or not, and in that item there are several alternatives ("quiero" or "deseo", etc.)

3.4 Language identification module

Our system needs to be multilingual. To this end, we have to detect the language using a short segment of speech and then switch to the speech recognizer specific of that language.

Different techniques can be used. In our group, we have used PPRLM, based in the modeling of the sequence of phonemes obtained for each of the languages considered using a very simple phoneme recognizer. More details and results can be read in [7].

4. Limitations of VoiceXML

VoiceXML is a very powerful language for dialog handling, and offers a great variety of control possibilities over the recognition and synthesis services. However, it offers so many possibilities that is quite difficult to support all the services offered. That is why most VoiceXML platforms available nowadays (BeVOCAL Cafe, Hey Anita, etc.) do not support all the functionalities.

Nevertheless, we have detected several limitations in VoiceXML definition that we will describe (see also [8]):

- In VoiceXML there is no specification regarding how to identify the different active recognizers. This issue can be solved using the <property> tag. Many things can be done using this tag, but none of them refers to the use of different recognizers in runtime. See in Section 2.1 how we have used this tag.
- The telephone control aspects are not completely defined. The consequence is that many VoiceXML platform developers do not implement the telephone part of the platform. Moreover, in the VoiceXML and OpenVXI mailing list [9] there is very little interest in the telephone aspects of VoiceXML.
- Handling of calls to returning dialogues. VoiceXML offers a mechanism to handle them, but it is only allowed in a <form>. We think that calling subdialogs from within <fields> or <blocks> is highly desirable.
- Compared to other programming languages, VoiceXML lacks common constructs for program logic, like loops (e.g. while and for). Generally, the usage of ECMA script is recommended. But this solution is useless when operations across fields or dialogues are needed.
- The access to external sources (e.g. a database) is only possible via CGI scripts. This is easy when transmitting data to a database. Nevertheless, problems arise when data from a database is to be returned into the dialogue flow. Here, VoiceXML code has to be generated by a CGI script that contains assignments of string constants to variables.
- To ease the automatic generation of VoiceXML scripts and simplify the handling of multilingual prompts it would be useful to establish the external representation of prompts by introducing prompt concepts.

5. Conclusions

We have developed a complete platform for dialog systems able to execute VoiceXML scripts using the functionality offered by OpenVXI. We have presented all the details necessary to adapt an existing system to the OpenVXI environment, highlighting and proposing solutions to the limitations that were found both in OpenVXI and in VoiceXML, which can be used by the standards associations.

We have also presented the most relevant modules of the platform, pointing out the use of standards in them, as SSML and JSGF.

6. REFERENCES

[1] Lehtinen, G., S. Safra, …, J.M. Pardo, R. Córdoba, R. San-Segundo, et al. 2000. "IDAS: Interactive Directory Assistance Service", VOTS-2000 Workshop, Belgium.

[2] R. Córdoba, et al. "An Interactive Directory Assistance Service for Spanish with Large-Vocabulary Recognition", Eurospeech 2001, pp. 1279-1282.

[3] Hamerich, S. W., V. Schubert, V. Schless, R. Córdoba, J.M. Pardo, L.F. d'Haro, B. Kladis, O. Kocsis, S. Igel. "Semi-Automatic Generation of Dialogue Applications in the GEMINI Project", Sigdial 2004.

[4] SpeechWorks web page: http://www.speechworks.com/.

[5] Pardo, J.M., et al. 1995. "Spanish text to speech: from prosody to acoustic". ICA, Vol. III.

[6] Burnett, D. C., M. R. Walker, A. Hunt. 2002. "Speech Synthesis Markup Language Version 1.0". W3C Working Draft, http://www.w3.org/TR/speech-synthesis.

[7] Córdoba, R., G. Prime, et al. 2003. "PPRLM Optimization for Language Identification in Air Traffic Control Tasks", EUROSPEECH, pp. 2685-2688.

[8] Hamerich, S. W., Y.F.H. Wang, V. Schubert, V. Schless, S. Igel. 2003. "XML-Based Dialogue Descriptions in the GEMINI Project". Proceedings of the "Berliner XML-Tage 2003", Berlín, Germany, pp. 404-412.

[9] VXIDiscuss. Mailing list for VoiceXML and OpenVXI interpreter: http://www.speechinfo.org/vxi-discuss/