# SPEECH INTERFACE IMPLEMENTATION FOR XML BROWSER

*Aki Teppo*

Helsinki University of Technology
Telecommunications Software and Multimedia Laboratory
P.O.Box 5400, FIN-02015 HUT, Finland
ateppo@cc.hut.fi

*Petri Vuorimaa*

Helsinki University of Technology
Telecommunications Software and Multimedia Laboratory
P.O.Box 5400, FIN-02015 HUT, Finland
Petri.Vuorimaa@hut.fi

## ABSTRACT

The growing popularity of digital cellular phones and personal digital assistants (PDA) is setting new demands for Internet content producers. One problem with these devices is the small visual display. WWW pages are usually designed for traditional desktop computers and they are difficult to view with small displays. In this paper, a solution is presented that uses the audio capabilities of such mobile devices in addition to optional visual display. The idea is to transform XML data into VoiceXML in addition to some traditional display layout language. This approach could also make Internet browsing possible for visually handicapped people.

## 1. INTRODUCTION

The paper describes how XML [1] data can be displayed and navigated through a speech interface in a consistent manner. This is done by first transforming the original XML data into VoiceXML [2] document and then using a VoiceXML capable browser to display it. This kind of speech interface can exist alone or side by side with a traditional browser interface. The coexistence of two interfaces is suitable for PDA environments, for example. The small visual display of PDA can be used to present images and the auditory display can be used to present textual information and to offer navigation capabilities. A demonstration is presented in the paper. The example consists of a small movie database, which is presented simultaneously through visual and auditory displays.

## 2. TECHNICAL BACKGROUND

In this chapter, two main technical topics are briefly explained. The first subject consists of data presentation and transformation related technologies, especially XML and its subsidiary technologies. The second topic consists of speech technology related technologies and products used in our research work.

### 2.1. XML Technologies

In short, XML is a method for presenting structured data in a text file and it looks a bit like HTML [3]. XML is a meta-language, which is used to describe other languages. Some of the design goals for XML are that it should be easy to write programs which process XML documents and that it should be straightforwardly usable over the Internet [1]. One important property of XML from our perspective is that XML documents must be well formed and no exceptions to this are allowed. This property means that the structure of data is unambiguous and this is useful when the data is interpreted and transformed into another XML language.

XSLT is a XML language used to describe transformations from one XML data tree into another XML data tree [4]. A XSLT document is called a stylesheet. Stylesheet contains template rules. A template rule contains a pattern, which is matched to the source XML data tree. A basic case is that this pattern consists of XML element identification (tag). The rule contains also the content, which replaces the original matched pattern in the result tree.

XSL FO is the formatting part of eXtensible Stylesheet Language (XSL) [5]. It contains formatting elements for pagination, layout and styling of documents. A possible publishing process is the following [5]: The original XML data is just structured data, it does not contain any formatting elements. Then XSLT is used to create the presentation of the original data and XSL FO elements are used to describe the styling. The resulting tree contains XSL FO elements with the original data elements.

X-Smiles [6] is a XML browser currently being developed in our laboratory at Helsinki University of Technology. It is implemented using the Java language. It currently supports several XML specifications including XSLT and XSL FO. It has its own API for supporting third party extension components (Markup Language Functional Component, MLFC). This API was used in our research to create a MLFC to support VoiceXML content. The browser will be published soon as open source software and it is going to be available at <http://www.x-smiles.org>.

### 2.2. Speech Interface

VoiceXML is designed for creating audio dialogs that feature synthesised speech, digitised audio, recognition of spoken and DTMF key input, recording of spoken input, telephony, and mixed-initiative conversations [2]. VoiceXML's main goal is to bring the full power of web development and content delivery to voice response applications, and to free the authors of such applications from low-level programming and resource management [2]. The language separates application logic from platform dependent details. This means that the same voice application can be easily deployed to different locations provided that the underlying platforms conform to VoiceXML specification.

The Java Speech API defines a cross-platform software interface to speech technology. Two core speech technologies are supported through the Java Speech API: speech recognition and speech synthesis. Speech recognition provides computers with the ability to listen to spoken language and to determine what has been said. In other words, it processes audio input containing speech by converting it to text. Speech synthesis provides the reverse process of producing synthetic speech from text. It is often referred to as text-to-speech technology. [7]

The speech synthesis system used in our research was Festival [8], which offers a general framework for building speech synthesis systems. It also includes some support for Java Speech API.

The speech recognition system used was Sphinx [9]. Sphinx is mainly a library, not an independent product. So a custom server application was developed during our research.

## 3. IMPLEMENTATION

The goal of the research project was to develop a system demonstrating the possibilities of VoiceXML combined with a more traditional browser interface. In this chapter, the resulting system is described. First, the overall architecture is presented and then the sub-components are presented individually. Finally the actual demonstration with example data is presented with some screenshots.

### 3.1. Overall Architecture

The demonstration software consists of three main parts. The first part is the X-Smiles browser package. The second one is the VoiceXML interpreter package and the third part is the engine package. The overall architecture is shown in Figure 1. The figure depicts also the relations between different packages.

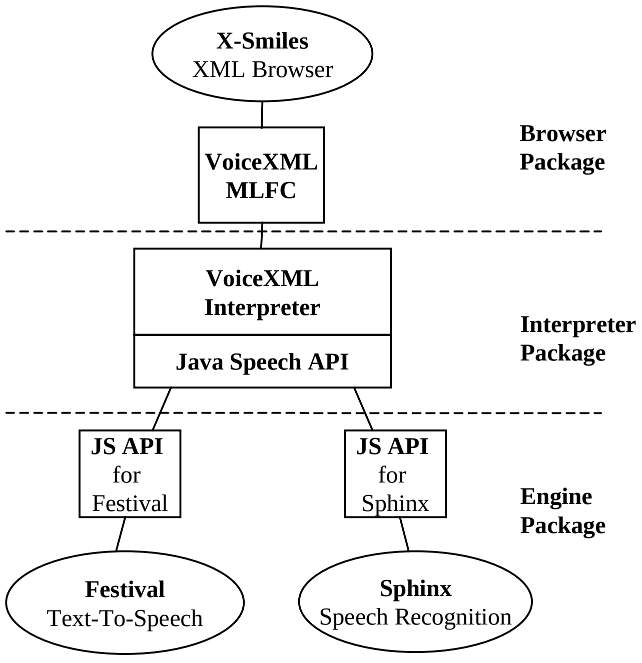


Figure 1. *The overall architecture.*

The browser package includes the X-Smiles browser and the VoiceXML Markup Language Functional Component (MLFC), which was implemented in the research project. The essential task of this package is to fetch the requested XML document and to pass it to appropriate MLFC. In our demonstration, the XML file contains XSL FO data, which is mediated to FO MLFC. The XML file also contains a reference to a VoiceXML data file, which is in turn passed to the VoiceXML MLFC.

The interpreter package contains a basic implementation for VoiceXML interpreter (actually VoiceXML interpreter could be called VoiceXML browser, but here it is called interpreter to avoid a possible mix-up with the X-Smiles browser). The main task of this package is to translate the VoiceXML content into suitable actions for underlying speech engines. These actions are defined in terms of Java Speech API so this interpreter is independent of the underlying speech engines.

The engine package contains engines for speech recognition and speech output. Also, the Java Speech API implementations for these engines are included here.

### 3.2. Simple VoiceXML Interpreter

An adequate VoiceXML interpreter for our purposes was implemented during the project. It covers only a small subset of the VoiceXML specification [2]. The interpreter was implemented using the Java language. All interaction between the interpreter and the underlying speech engines was implemented using the Java Speech API (JSAPI) [6]. The main advantage of using the JSAPI is that the interpreter is independent from the underlying engines. This means that the underlying engines can be changed with minimal effort given that the new engines support JSAPI.

The goal of the speech interface was to be able to output the document contents and to offer basic navigation capabilities, so the most relevant VoiceXML elements for our purposes were the “prompt” and the “menu” element. The “prompt” element is used for speech output and the “menu” element is used for navigation. The “form” element was completely ignored, because the interface did not need to gather any data input from the user in addition to the navigation choices.

### 3.3. Java Speech API implementation for Festival and Sphinx

The Text-To-Speech (TTS) engine used was Festival [8]. Festival includes quite good support for Java Speech API. However, because of licensing reasons the Festival group could not release one library needed for the JSAPI support [10]. This utility library from Sun Microsystems contained some basic implementation for JSAPI. So, the functionality of this utility library had to be implemented. As a result an adequate JSAPI library was created and it was also used with the JSAPI support for Sphinx.

As Sphinx is an Automatic Speech Recognition (ASR) library, the actual ASR engine application had to be implemented. The Sphinx library is written in C language and it can be compiled in Unix environments. Our development environment was Linux (RedHat 6.2). The Sphinx package contains example client/server software, which was used as a basis for our JSAPI server. One compromise had to be made during development: The resulting server does not support dynamic grammar loading, so it has to be started again every time the grammar changes. This leads to a longer delay between grammar changes.

The Java part of JSAPI support was implemented to fulfill our needs. The complete JSAPI support would have been quite a large task, so only the required parts were implemented. The Java part of JSAPI connects to ASR server through TCP/IP, which is used to transfer control signals.

### 3.4. Integration to the X-Smiles Browser

As mentioned in chapter 3.1 the X-Smiles browser has a kind of plug-in interface for third party XML interpreters, MLFCs. This

API was used to create the VoiceXML MLFC. The implementation was quite simple, because the MLFC acts merely as a wrapper to the VoiceXML interpreter discussed in chapter 3.2.

X-Smiles already included a XSL FO interpreter, which was used as a visual display for the XML data. This MLFC also supports secondary MLFCs. It means that there can be two XML interpreters at the same time. This feature was used in our demonstration in the following way: The VoiceXML part was linked to the FO document as an "external-graphic" element. Now, when FO MLFC tries to load this data and notices it is in some other XML language, a secondary MLFC is loaded. Naturally, in our case this secondary MLFC is a VoiceXML MLFC.

### 3.5. Transforming the Sample XML Data

The sample XML data was extracted from [11]. The data contains information related to movies. Figure 2 shows the information of one movie. The goal was to render the relevant parts of the data through TTS engine and to provide navigation through speech recognition engine.

```
<movie name="Star Wars" id="star">
       <information>
         When the opening scroll of Star Wars
         mentions "a galaxy far, far away,"
         it might unwittingly refer to the
         '70s, a time when "the force" went
         hand in hand with "the Fonz," and
         hokeyness ran unchecked.
       </information>
       <picture file="sw.jpg"/>
</movie>
```

Figure 2. *Extract from the sample data.*

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>

<xsl:template match="/">
<vxml version="1.0">
<xsl:apply-templates select="movies"/>
</vxml>
</xsl:template>

<xsl:template match="movies">
<!-- Creates the main menu -->
<!-- Writes the movie specific information into
separate file using Xalan extensions -->
</xsl:template>

</xsl:stylesheet>
```

Figure 3. *XSLT to transform sample data to VoiceXML.*

The presentation chosen was a two-level menu. In the main menu all the movies are offered and when the user chooses one of them, the related information is displayed. A natural way was to collect all the "name" attributes of the "movie" elements and create a VoiceXML menu of them. The "information" elements of individual movies were then collected into separate files. The developed XSL Transformation stylesheet is shown in Figure 3 and the resulting VoiceXML menu is shown in Figure 4.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE vxml SYSTEM "voicexml1-0.dtd">
<vxml version="1.0">
<menu>
 <prompt>
   Welcome to current movies
 </prompt>
 <prompt>
   Please, select one of: <enumerate/>
 </prompt>
 <choice next="pulp.fo">Pulp Fiction</choice>
 <choice next="fifth.fo">Fifth Element</choice>
 <choice next="star.fo">Star Wars</choice>
 <choice next="psycho.fo">Psycho</choice>
 <choice next="mulan.fo">Mulan</choice>
 <choice next="sound.fo">Sound of Music</choice>
</menu>
</vxml>
```

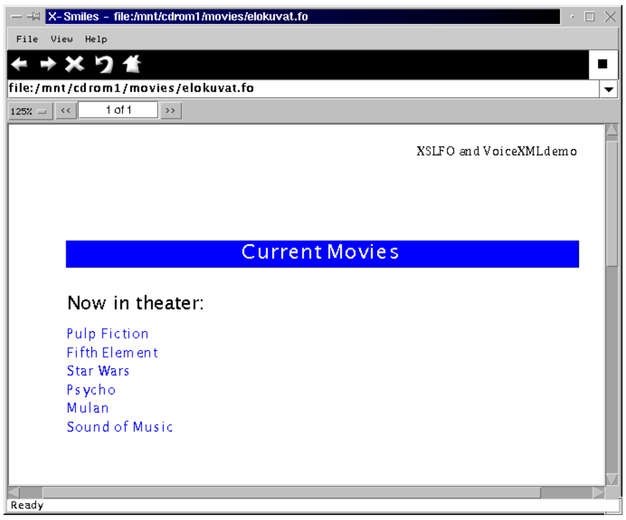
Figure 4. *The resulting VoiceXML main menu.*

Similar transformation process was also created to produce the XSL Formatting Objects version of the data.

## 4. RESULTS

In this chapter, the results of development are represented. First the system functionality is represented by means of the example data and then the system performance is discussed. The example data and transformations related to it were already described in section 3.5.

The original data was transformed into two XML languages: XSL FO and VoiceXML. XSL FO was used to define the layout of visual display and VoiceXML was used to define the speech interface dialog. The resulting visual main menu is shown in Figure 5. From the main menu user can choose a movie and see the details of it. The detail page of one movie is shown in Figure 6.



X-Smiles - file:/mnt/cdrom1/movies/elokuvat.fo

file:/mnt/cdrom1/movies/elokuvat.fo

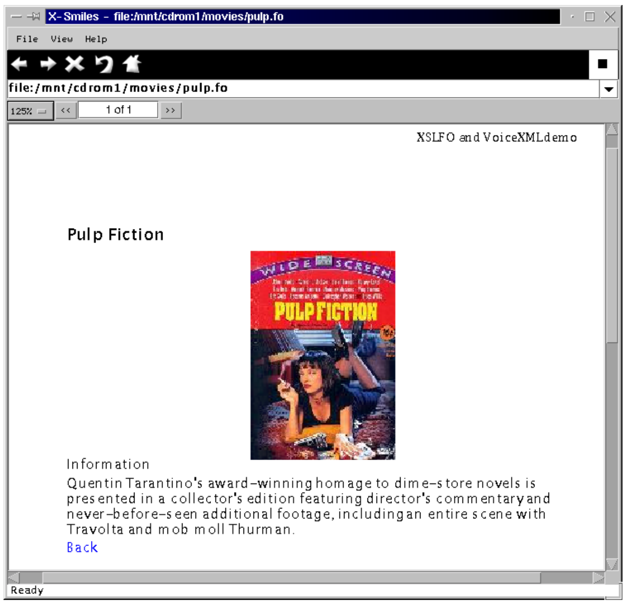
XSLFO and VoiceXMLdemo

Current Movies

Now in theater:

Pulp Fiction
Fifth Element
Star Wars
Psycho
Mulan
Sound of Music

Figure 5. *The XSL FO main menu of the demonstration.*



file:/mnt/cdrom1/movies/pulp.fo

XSLFO and VoiceXMLdemo

Pulp Fiction

Information

Quentin Tarantino's award-winning homage to dime-store novels is presented in a collector's edition featuring director's commentary and never-before-seen additional footage, including an entire scene with Travolta and mob moll Thurman.

Back

Figure 6. *Individual movie page after choosing "Pulp Fiction" from the main menu.*

When each of these pages opens, the contents are also rendered through the speech interface. User can navigate with mouse or through the speech recognition interface. A sample voice interface dialog is represented in Figure 7.

```
Browser: Welcome to current movies! Please
select one of: Pulp Fiction, Fifth Element, Star
Wars, Psycho, Mulan, Sound Of Music.
User: Pulp Fiction
Browser: Pulp Fiction – Information – Quentin
Tarantino's award-winning homage to dime-store
novels is presented in a collector's . . .
Please select one of: Back
User: Back
Browser: Welcome to current movies! . . .
```

Figure 7. *A sample VoiceXML dialog.*

The system was tested with two different configurations. The first configuration was Intel Pentium II 300MHz with RedHat Linux 6.2 and 64Mb memory. The second configuration was Intel Celeron 450MHz with RedHat Linux 6.1 and 128Mb memory. Frankly speaking, the first configuration was completely unusable. This is due to the high memory usage of the used TTS and ASR engines. When opening the main menu it took minutes before the TTS started to render the text and the ASR was ready to get response from the user. However, the second configuration was much more usable. TTS started in a few seconds and speech recognizer was ready in about ten seconds after opening a page.

## 5. CONCLUSIONS

In this paper, the sample data was a custom XML language file, which was transformed into two presentation languages. XSL Formatting Objects was used for visual and VoiceXML for auditory display. XSL Transformation was used as a tool for converting the original data into these presentation formats. This tool suited well for the task and more complicated transforms could be possible. One could maybe use XSLT to transform more common visual layout languages, like XHTML [12], into VoiceXML. These kinds of transforms from huge amount of existing content into VoiceXML would make the content creation easier for speech interfaces. Naturally, VoiceXML is not enough alone. A lot depends on the speech engine manufacturers, namely whether they are going to support VoiceXML extensively or not.

## 6. REFERENCES

[1] T. Bray et al., *Extensible Markup Language (XML) 1.0 (Second Edition)*, <URL: http://www.w3.org/TR/2000/REC-xml-20001006>, 2000.

[2] VoiceXML Forum, *Voice eXtensible Markup Language VoiceXML*, <URL: http://www.voicexml.org>, 2000.

[3] B. Bos, "XML in 10 points", <URL: http://www.w3.org/XML/1999/XML-in-10-points>, 1999.

[4] J. Clark, *XSL Transformations (XSLT) Version 1.0*, <URL: http://www.w3.org/TR/xslt>, 1999.

[5] W3C, *Extensible Stylesheet Language (XSL) Version 1.0*, <URL:http://www.w3.org/TR/2000/CR-xsl-20001121>, pp. 1-6, 2000.

[6] P. Vuorimaa, T. Ropponen and N. von Knorring, "X-Smiles XML Browser", the 2nd International Workshop on Networked Appliances, IWNA'2000, New Brunswick, NJ, USA, Nov. 30 - Dec. 1, 2000.

[7] Sun Microsystems, *Java Speech API Programmer's Guide version 1.0*, pp. 1, 1998.

[8] A.W. Black, P. Taylor, and R. Caley, *The Festival Speech Synthesis System - System documentation*, <URL: http://www.cstr.ed.ac.uk/projects/festival/manual/festival_toc.html>, 1999.

[9] K-F. Lee, H-W. Hon, and R. Reddy, "An Overview of the SPHINX Speech Recognition System", IEEE Transactions on Acoustics, Speech and Signal Processing, vol. 38, no. 1, Jan. 1990.

[10] Festival distribution version 1.4.0, file: festival/src/modules/java/cstr/festival/jsapi/ReadMe, 1999.

[11] O. Marttila and P. Vuorimaa, "XML Based Mobile Services", the 8th Int. Conf. in Central Europe on Computer Graphics, Visualization, and Interactive Digital Media, WSCG'2000, Czech Republic, Feb. 7-10, 2000.

[12] W3C, *XHTML™ 1.0: The Extensible HyperText Markup Language*, <URL: http://www.w3.org/TR/2000/REC-xhtml1-20000126>, 2000.