Royal Institute of Technology, KTH
IMIT

Practical Voice over IP
2G1325

VoiceXML

For: Professor Gerald Q. Maguire Jr.
By: Andreas Ångström, it00_aan@it.kth.se and
Johan Sverin, it00_jsv@it.kth.se
Date: 2004-05-24

Abstract

Today VoiceXML is the standard scripting language for rendering web pages over the telephone. Developing costs of an interactive phone application have changed dramatically with the new markup language.

VoiceXML builds on the basic concept and rules set by XML. Interactive applications contain synthesized speech, pre-recorded audio, grammars defining words that could be recognised, and DTMF key input. By saying something or pressing the keypad on the phone the user transitions between different pages.

VoiceXML could be used in many different ways. You can integrate it with your web page, letting people access it through a phone. It is simple to create services such as booking a ticket or looking up when the bus leaves. It can be used to create a voicemail from your phone or having your regular e-mail be read to you. Some vendors of voice gateways even offer SMS add-ons.

The neat thing is that VoiceXML works both with the traditional PSTN system and the new technology of Voice over IP (VoIP).

This paper describes how VoiceXML is defined and how it works. It also contains a short discussion between the differences in SALT, Microsoft's voice language, and VoiceXML. We have tried to implement our own service and it is described in detail in chapter 6.

Our conclusion is that VoiceXML is a great way to create voice-enabled applications. However the Text-To-Speech and speech recognition engines on the market have to be improved before they will be convenient for regular users to use.

This report was written by Andreas Ångström (chapters 3-8), and Johan Sverin (chapters 1-2 and the coding of our project).

Table of contents

1 Introduction

Voice eXtensible Markup Language (VoiceXML) is a web-based language designed to easily, and at a low cost, create voice-enabled applications. Together with digitised audio, synthesized speech, recognition of spoken and DTMF key input, recording of spoken input and telephony this web-based development has become a popular way to create voice-enabled applications. While HTML was designed for visual web sites, VoiceXML was designed to create a dialog between a human and an application. You can look at the VoiceXML-interpreter like a voice browser.

The language allows a developer to easily create voice-enabled services such as voicemail and information portals. It can be integrated with your web page, enabling people to access it through the phone. As the voice gateway uses VoIP protocols to communicate with the voice browser, it does not matter whether the user calls from the regular PSTN system or an IP network.

1.1 Brief history

It all started with a research project called PhoneWeb at AT&T Bell Laboratories. This became the basis of VoiceXML [1]. The project's goal was to create a development tool that combined traditional telephony services with the web based services. Another important issue was that it should also be easy for a person that was not directly involved with the development of the technique to create new services. Later, the company was reorganized into AT&T Labs and Lucent's Bell Labs and they both continued to develop independent versions of their markup languages.

There were two other companies that also should be considered as pioneers; IBM's SpeechML and Motorola's VoxML. These four companies understood that they had to come up with a standard and together they created a forum called VoiceXML. They chose XML as the basis for this effort, because they thought that this was the direction that web technology was going [2]. The forum grew and in March 2000, VoiceXML version 1.0 was released.

Today, four years later, VoiceXML is the standard scripting language for rendering Web pages over the telephone [2]. In March this year (2004), the World Wide Web Consortium (W3C) has advanced VoiceXML 2.0 to the recommendation status, and thereby established as a Web Standard.

1.2 eXtensible Markup Language (XML)

Today, XML has become the de facto standard for defining structured documents and data on the web [4]. VoiceXML builds upon this basic concept and the rules set forth by XML.

An XML document consists of one or more named elements that are organized in a nested hierarchical way. An element begins with an opening tag (the less-than symbol, <), some data, and a closing tag (the greater-than symbol, >). For any given element, the name of the opening tag must match that of the closing tag. A closing tag is identical to the opening tag except that the less-than symbol is immediately followed by a forward-slash (/). You should also have in mind that the element names are case sensitive.

```
<prompt>Please choose a language</prompt>
```

By placing a forward-slash prior to the greater-than symbol, an opening tag can be combined with a closing tag. This is useful if an element does not contain any data.

```
<break/>
```

Each element may also have attributes. An attribute specifies which properties the element has and consists of a name/value pair. The value must be contained in matching single or double quotes.

```
<field id=”lang”>
   <prompt>Please choose a language</prompt>
   <break time=“500ms“/>
</field>
```

All elements can be rearranged in an infinitely nested hierarchy, but only one element can be the root document element. The root document element is the element that appears first in the document. With some exceptions, all other elements must be children of the root element.

2 VoiceXML 2.0

2.1 How it works

Most of our current telephones, including our cell phones, do not have enough processing power, memory, or battery resources to support a voice browser [11]. Therefore, the voice browser resides on the network in a speech server. The speech server can be located at large companies or off-site at a hosting server.

It all starts with a user that places a telephone call from any phone using the PSTN. The company’s PBX (private branch exchange) routes or forwards the call to the VoIP Gateway. The gateway acts as a media translator or protocol converter. The voice server is the component that receives the call from the gateway, via the VoIP protocol. When you configure your voice server, you configure both the telephone numbers to respond to and the response; that is, you determine what files to load for each telephone number on the list [24].

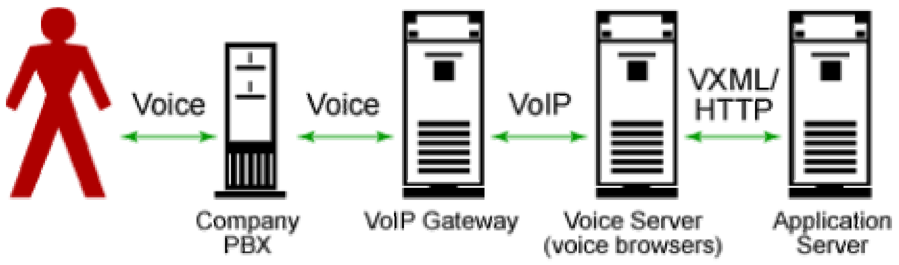

Figure 1, Infrastructure, [24]

The voice browser downloads VoiceXML scripts, grammar formats, and audio files that are needed from the application server. The voice browser can then prompt a menu to the user, by relaying pre-recorded audio or synthesizing speech. The VoIP streams are converted from TCP/IP packets to voice before sending them out into the telephone network. The user responds either by speaking or pressing the buttons on a touchtone telephone. The speech and tones from the telephone travels through the PSTN and are processed by the voice server. The voice browser can then access different applications or databases via the Internet [6].

VoiceXML is a language that: [5].

- Minimizes client/server interactions by specifying multiple interactions per document.
- Shields application authors from low-level and platform-specific details.
- Separates user interaction code (in VoiceXML) from service logic (e.g. CGI scripts).
- Promotes service portability across implementation platforms. VoiceXML is a common language for content providers, tool providers, and platform providers.
- Is easy to use for simple interactions, and yet provides language features to support complex dialogs.

2.2 Concepts

A single VoiceXML document, or a set of documents, called an application, forms a conversational finite state machine. The user can only be in one state, or dialog, at a time. Each dialog contains information about the transition to the next dialog. The transitions are specified by URI's. The execution of an application ends when a dialog does not specify a successor, or if it has an element that explicitly exits the application.

2.2.1 Dialogs and Subdialogs

VoiceXML has two types of dialogs; forms and menus. The difference between them is that the form defines an interaction that collects values for a set of form item variables, while a menu presents the user with some option and goes to other dialogs based on a choice.

A subdialog can be viewed as a function call. It provides a mechanism to invoke a new interaction, and then returning to the original form. Variable instances, grammars, and state information are saved and are available upon returning to the calling document. Subdialogs can be used to create a confirmation sequence that may require a database query; to create a set of components that may be shared among documents in a single application; or to create a reusable library of dialogs shared among many applications. You will see how this works later in chapter 6.

2.2.2 Sessions

A session begins when the user starts to interact with the VoiceXML interpreter context, and ends by a request from the user, a document, or the interpreter context. New documents, grammars and audio files are loaded and processed while the session is still alive.

2.2.3 Applications

An application is a set of documents that share the same application root document. When a document in the application is loaded, so is the application root document. The application root document remains loaded while transitioning between documents in the same application. First when a document that is not in the application is loaded, the application root document is unloaded. All variables in the application root document are accessible from any other document in the application as application variables. The grammars loaded by the root document are also active until it is unloaded.

Figure 2 shows how documents (D) share the same application root document (root). The root document is loaded while transitioning between the other documents.

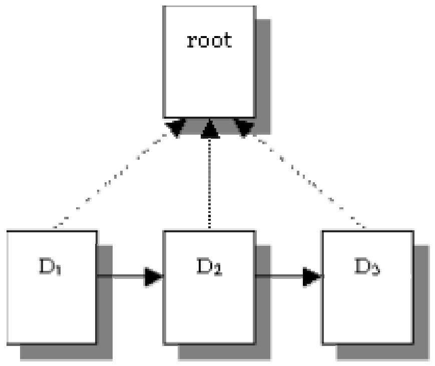

Figure 2, Transitioning between documents in an application, [5]

2.2.4 Grammars

Each dialog has one or more speech and/or DTMF grammar associated with it. There exist two types of grammars in VoiceXML; machine directed and mixed initiative. Mixed initiative adds more flexibility and power to the application because the grammars are not active in just one dialog, like in the case of machine directed. With a mixed initiative grammar active, a user can say something matching another dialog's active grammar, and the execution transitions to that dialog.

2.2.5 Events

Events are thrown in a large variety of circumstances. It can happen when a user does not respond, or when a user does not respond intelligibly. The interpreter can also throw an event when it finds a semantic error in a VoiceXML document. All events are caught by <catch> elements or by their syntactic shorthand.

2.3 How to build an application

When building a voice application, it is useful to define resources like variables, grammars, scripts, and event handlers that can be shared across the application. Variables declared in the application root document allow you to share state across documents that make up your application. Having scripts in your application root document allows you to load the script once for your application. Grammars and event handlers declared in the application root document are active throughout the application.

2.3.1 Forms

Forms are one of the most important components in VoiceXML. VoiceXML forms are analogous to web forms. You use them to collect information (voice input) from the user. A form contains form items, elements that are used by the main loop of the form interpretation algorithm (FIA). The form items can be divided into two different types, input items that can be "filled" by the user and control items that cannot. The form also contains declarations of non-form item variables, event handlers, and "filled" actions. The <filled> element specifies an action to perform when some combination of input items are assigned.

2.3.1.1 Form Interpretation Algorithm (FIA)

A special algorithm, the form interpretation algorithm (FIA), is used to interpret forms. The algorithm has a main loop that repeatedly selects a form item and visits it. The next unfilled form item is the one that is selected. When a form item is visited, one or more prompts are selected and then played. Then the algorithm collects the user's input. This can be done by either filling in some input items or by throwing an event (help, for instance). Any <filled> actions that pertained to the newly filled input items are then interpreted. The FIA ends when it interprets a transfer control of statement (e.g. a <goto> to another dialog or document, or a <submit> of data to the document server). It is also possible that it ends with an implied <exit> when no form item remains intelligibly to select.

2.3.1.2 Input Items

Input items have prompts to inform the user what to say or key in, grammars to define what input is allowed and event handlers that take care of different events that can occur. The input items defined in VoiceXML version 2 are:

<field>	An input item whose value is obtained via Automated Speech Recognition or DTMF grammars.
<record>	An input item whose value is an audio clip recorded by the user.
<transfer>	An input item which transfers the user to another telephone number. If the transfer returns control, the field variable will be set to the result status.
<object>	This input item invokes a platform-specific "object" with various parameters. The result of the platform object is an ECMAScript Object.
<subdialog>	A <subdialog> input item is roughly like a function call. It invokes another dialog on the current page, or invokes another VoiceXML document. It returns an ECMAScript Object as its result.

Table 1, Input items, [5]

2.3.1.3 Control Items

There are two types of control items:

<block>	A sequence of procedural statements used for prompting and computation, but not for gathering input.
<initial>	This element controls the initial interaction in a mixed initiative form. Its prompts should be written to encourage the user to say something matching a form level grammar.

Table 2, Control items, [5]

2.3.1.4 Directed Forms

The simplest type of form is one in which form items are executed once and in a sequential order. Here is a short example of such a service.

```
<form id="weather_info">
   <block>Welcome to the weather information service.</block>
   <field name="state">
      <prompt>What state?</prompt>
      <grammar src="state.grxml"  type="application/srgs+xml"/>
   </field>
   <field name="city">
      <prompt>What city?</prompt>
      <grammar src="city.grxml" type="application/srgs+xml"/>
   </field>
   <block>
      <submit next="/servlet/weather" namelist="city state"/>
   </block>
</form>
```

Figure 3, Directed form, [5]

Each field in this example has a prompt to play and a grammar that specifies what to listen for. The answers are submitted to the servlet.

2.3.1.5 Mixed Initiative Forms

Mixed initiative forms can be used to collect multiple pieces of information in a single call state. This allows the flow of the call to be directed by the user and by the application.

The <initial> element is used to prompt the user to participate in the mixed initiative dialog [18]. It is important that the VoiceXML interpreter executes it before any other field and that it does not define any grammars. However, the <initial> element can contain elements such <noinput>, <nomatch>, <help>. This helps the user to complete the dialog.

If the user does not provide all information prompted by the <initial> element, the voice interpreter executes the content of individual fields, to collect the missing parts. Say you have a service where the user is prompted to say starting and destination cities, but the user only says “to Stockholm, Sweden”. To help the user complete the dialog you should define a <field> element named “from”. You should also define a <field> element named “to”, if the user only provides information about the starting city.

Mixed initiative forms can be difficult to understand, but are very useful and the forms give a better flow.

2.3.2 Menus

A menu could be seen as a form containing a single anonymous field that prompts the user to make a choice, and then based on that choice makes a transition to another dialog or document. The following menu offers the user two choices:

```
<menu>
   <prompt>Welcome. Say one of: <enumerate/></prompt>

   <choice next="news.vxml">News</choice>
   <choice next="sports.vxml">Sports</choice>

   <noinput>Please say one of <enumerate/></noinput>
</menu>
```

2.3.2.1 DTMF in menus

You can use the <menu>’s dtmf attribute to allow a choice of responding by either voice or by pressing the buttons on the keypad. When set to true, the first nine choices that have not explicitly specified a value for the dtmf attribute are given the implicit ones “1”, “2”, etc. An explicitly specified value is a value that has been manually assigned, for example dtmf=”12”. Remaining choices will not be assigned a dtmf value. You can also use DTMF by including a <property> element in the menu.

Here is an example of the same menu as above, but with only DTMF:

```
<menu>
   <property name=”inputmodes” value=”dtmf” />
   <prompt>Welcome. For news press one, for sports press two:</prompt>

   <choice dtmf=”1” next="news.vxml">News</choice>
   <choice dtmf=”2” next="sports.vxml">Sports</choice>

   <noinput> For news press one, for sports press two:</noinput>
</menu>
```

2.3.2.2 The enumerate element
The <enumerate> element generates the description of available choices automatically. It specifies a template that is applied to each choice in the order they appear in the menu. If it is used without any content, a default template lists all choices one by one. If it has content, the content is the template specifier. The specifier refers to two special variables: _prompt is the choice’s prompt, and _dtmf is a normalized presentation of the choice’s assigned DTMF sequence. If the example was rewritten as:

```
<menu dtmf=”true”>
   <prompt>Welcome.
   <enumerate>
      For <value expr=”_prompt”>, press <value expr=”_dtmf”>.
   </enumerate>
   </prompt>

   <choice dtmf=”1” next="news.vxml">News</choice>
   <choice dtmf=”2” next="sports.vxml">Sports</choice>

</menu>
```

Then the menu prompt would be:
C: Welcome. For News, press 1. For Sports, press 2.

2.3.3 Document head

Below is a short example of what a well-defined VoiceXML document may look like. This example does not require any input from the user.

```
<?xml version ="1.0"?>
<vxml version="2.0" xmlns=http://www.w3.org/2001/vxml">
   <form>
      <block>Welcome to KTH, Sweden.</block>
   </form>
</vxml>
```

The first tag <?xml> tells the interpreter that this is an xml document. The next element <vxml>, is the top-level element, which mainly is a container for dialogs. The <form> element defines the type of dialog that should be used. Here is a form used, that presents and gathers information (see section 2.2.1 Dialogs and subdialogs). This form only contains a <block> that synthesizes audio and presents “Welcome to KTH, Sweden”.

version	The version of VoiceXML of this document (required). The current version number is 2.0.
xmlns	The designated namespace for VoiceXML (required). The namespace for VoiceXML is defined to be http://www.w3.org/2001/vxml.
xml:base	The base URI for this document as defined in [XML-BASE]. As in [HTML], a URI which all relative references within the document take as their base.
xml:lang	The language identifier for this document. If omitted, the value is a platform-specific default.
application	The URI of this document's application root document, if any.

Table 3, <vxml> attributes, [5]

2.3.4 Interaction example

In the next example we create an interaction between the user and the application. First the application gives the user some available options. The result is then sent to a server script where the user's answer is analysed. A <field> element is an input-field that requires an input before moving on in the form.

```
<?xml version ="1.0"?>
<vxml version="2.0" xmlns=http://www.w3.org/2001/vxml">
   <form>
      <block>Welcome to KTH, Sweden.</block>
      <field name=”answer”>
         <prompt>Would you like news or information about courses?</prompt>
         <grammar src=”main_option.grxml” type=”application/sgrs+xml” />
      </field>
      <block>
         <submit next=”analyze.asp” namelist=”answer” />
      </block>
   </form>
</vxml>
```

An example of how an interaction of the above application can be is as follows.

C: Welcome to KTH, Sweden
C: Would you like news or information about courses?
U: Schedule
C: I did not understand what you said (default answer)
C: Would you like news or information about courses?
U: Information about courses

The dialog continues in “analyze.asp”.

The information that the user gives to the application does not have to match the whole phrase. The grammar defines what combinations of words and phrases are allowed.

2.4 Speech grammar

A grammar in VoiceXML is specified by the <grammar> element. The speech grammar specifies a set of utterances that a user may speak to perform an action. VoiceXML platforms must support at least one common format, the XML Form of the W3C Speech Recognition Grammar Specification (SRGS). VoiceXML platforms should also support the Augmented BNF (ABNF) Form of the W3C Speech Recognition Grammar Specification (SRGS) [5].

There are two types of grammar: inline grammar and external grammar. An inline grammar is specified by the content of the <grammar> element and defines an entire grammar.

```
<grammar type=”media-type” mode=”voice”>
   inline speech grammar
</grammar>
```

An external grammar is specified by an element of form:

```
<grammar src=”URI” type=”media-type” />
```

For an inline grammar the type parameter specifies a media-type that governs the interpretation of the <grammar> element.

Grammars match spoken words or touch-tones. These words are referenced to as tokens [13]. The simplest grammars that can be created are token strings composed only of one or more words. We can create a grammar that corresponds to a name.

```
<grammar>
   <item>Albert Einstein</item>
</grammar>
```

When creating a larger grammar, sub-grammars can be used to define re-usable grammar components. A sub-grammar is included into other grammars by a rule reference. These references can point to internal or external grammars.

A grammar file also consists of one or more grammar rules. An XML grammar defines rules using the <rule> element. Each rule has a unique name assigned to it, using the id attribute.

To extend our grammar we can create a name rule that includes references to a list of first and last names. The <one-of> element may contain one or more <item> elements, which contains a string to each alternate utterances.

```
<rule id=”Name”>
   <ruleref uri=”#firstName” />
   <ruleref uri=”#lastName” />
</rule>

<rule id=”firstName”>
   <one-of>
      <item>Albert</item>
      <item>Isaac</item>
   <one-of>
</rule>

<rule id=”lastName”>
```

```
   <one-of>
      <item>Einstein</item>
      <item>Newton</item>
   <one-of>
</rule>
```

This grammar matches four different utterances: Albert Einstein, Albert Newton, Isaac Einstein and Isaac Newton. Using the different operators that VoiceXML 2.0 supports allows a developer to create more complex word patterns. The XML grammars attribute “repeat” within an item element can be used to define optional rules and rule expansions [13].

2.5 Dual-Tone MultiFrequency (DTMF)

Dual-Tone MultiFrequency, or DTMF, is a method used in telephone systems to communicate with the keys that you used for dialling. Pressing a key on the phone’s keypad generates two tones, i.e. two sine waves, one for the row and one for the column [12]. These are decoded by the exchange to determine which key that was pressed.

This can easily be implemented in a VoiceXML application. When building up menus the developer can allow the user to respond by voice or by pressing a key, using DTMF (see section 2.3.2.1 DTMF in menus).

DTMF can also be used to gather input to forms together with voice. This example shows how a variable, PIN, is assigned a value by either using voice or pressing the keypad.

```
<form id="get_id">
   <field name="ID" type="digits">
      <prompt>Say or key in your personal identification number.</prompt>
   </field>
   <filled>
      <assign name="PIN" expr="ID" />
   </filled>
</form>
```

3 Text-To-Speech (TTS) synthesis

Text-To-Speech synthesis (TTS) gives your computer the possibility to convert arbitrary text into audible speech. One of the goals with TTS is to be able to provide textual information to people via voice messages [14]. TTS provides voice output for all kinds of information that could be stored in databases and information services. In the future, TTS can perhaps be used to listen to books being read.

Text-To-Speech quality is characterised by two factors; the intelligibility of the speech that is produced, and the naturalness of the overall message that is spoken [14]. Without a high intelligibility, a TTS system would be useless, and without naturalness the voice would be annoying to listen to.

3.1 Text-To-Speech conversion

A Text-To-Speech engine has different components that analyse the text and converts it into digitized audio.

3.1.1 Text normalisation

The text normalisation component of the engine takes the text input and breaks it into a series of spoken words [15]. A string like “Welcome to KTH” is broken up into “Welcome”, “to”, “KTH” along with a marker indicating that a period occurred.

First, text normalisation isolates words in the text. Then it searches for numbers, dates, and other symbolic representations that are analysed and converted into words. With help from a database the engine searches for abbreviations in the text. Other text transformations are defined by additional rules.

3.1.2 Homograph disambiguation

In English and many other languages, words can have different pronunciations, for instance the English word “read”, can be pronounced “reed” or “red”. A homograph is a word with the same text as another word, but with a different pronunciation.

Text-To-Speech engines use a variety of techniques to disambiguate the pronunciations. The most robust way is to try to figure out what the text is talking about and decide which meaning the given context has [15].

3.1.3 Word pronunciation

The pronunciation module accepts the text and outputs a sequence of phonemes, just like in an ordinary dictionary. When creating a Text-To-Speech engine, a pronunciation lexicon is also created. When a word comes into the module it tries to look it up in the lexicon. If that fails it has to try the “letter to sound” rules.

”Letter to sound” rules guess the pronunciation of the word of a text. There are number of techniques and algorithms describing how to do that, but that is outside the scope of this paper.

3.1.4 Prosody

Without prosody, synthetic audio would sound very robotic. Prosody defines the pitch, speed and volume that word, phrases, and sentences are spoken with. First, the engine identifies the beginning and ending of sentences, then pauses are placed between them.

The engine also identifies boundaries, such as noun phrases and verb phrases. Pauses are also placed between phrases and commas.

The words in a sentence that have an important meaning are emphasized. Words that are emphasized are louder, longer, and have more pitch variation. Non-important words, such as words that only make the sentence grammatically correct, are de-emphasized.

The output from the final prosody is simply a list of phonemes with pitch, duration, and volume.

3.1.5 Play audio

Now the synthesized speech is almost ready. The remaining step is to convert the list of phonemes into digitized audio. Many Text-To-Speech engines generate the audio by concatenating short recordings of phonemes [15]. In a simplistic form, the engine receives the phoneme, loads the digitized audio from a database, does some pitch, time, and volume changes, and sends it out to the sound card.

These recordings are made from a real person; a person reads from a carefully selected text that contains as many phonemes as possible. The database generally contains at least 1000-5000 different recordings.

4 Speech recognition

VoiceXML takes speech recognition further than before. Instead of talking to your computer, you are talking to a web site, and perhaps you are doing this through your phone. Speech recognition can be seen as speech-to-text, i.e. the process of converting speech into words, and is an important component for VoiceXML to work efficiently. It allows you to feed input to the browser with speech instead of clicking with the mouse, pressing the keyboard, or pressing the keypad on your phone.

The speech recognition engine is the component that processes the speech and tries to convert it into understandable words.

4.1 Terms and concepts

4.1.1 Utterance

When a user is saying something, this is known as utterance. An utterance is any stream of speech between periods of silence. All utterances are sent to the speech recognition engine. How does this work? Well, the speech recognition engine listens for speech input. When the engine detects audio input the beginning of an utterance is signalled. When a certain amount of silence is detected the end of the utterance occurs.

If the user does not say anything, the engine can detect this and send a silence timeout to the application. When the application receives this timeout it can take appropriate action.

An utterance can either be a single word or a whole phrase. By using a grammar the VoiceXML application can determine if the word or phrase is valid within a certain dialog. However, if the user spoke an entire phrase, but paused too long between the words the engine will think that the utterance has ended.

4.1.2 Pronunciations

Pronunciations are key components in speech recognition. A word’s pronunciation represents what the speech engine thinks that the word would “sound” like.

Words can have multiple pronunciations associated with them. For example, take the English word “the”. It can have pronunciations “thee” and “thuh” (see section 3.1.3 Word pronunciation)

4.1.3 Grammars

When creating a VoiceXML application the developer has to specify to the speech engine what words or phrases are allowed in the application. This is done by defining grammars. The engine compares the words in the grammars with the utterance from the user. If the user says something that is not in the grammar the engine would not be able to decipher it correctly.

4.2 Speech recognition in the telephony industry

VoiceXML is designed for speech recognition over the telephone, and this introduces some challenges. For instance, the bandwidth over the telephone is much lower than from a desktop. The traditional telephone system uses a sampling rate of 8 kHz, while the desktop uses 22 kHz [10]. This makes the speech recognition process much more difficult.

The telephone environment can also be very noisy. Users can be outside, in a car, at a mall or at an airport. Imagine the challenge if the user is driving on a highway, with the windows down, and the stereo pumping at the highest volume.

In a noisy environment people tend to speak louder, which is not a help for the recognition engine. On the contrary, this distorts the voice and hinders the speech recognition process. This phenomenon is known as "Lombard Speech" [10].

5 VoiceXML versus SALT

Speech Application Language Tags (SALT) is Microsoft's response to VoiceXML. SALT extends existing mark-up languages such as HTML, XHTML, and XML. It is not yet standardised, but W3C has the specification under consideration. However, the language has support from many big companies including Cisco Systems, Converse, Intel, Microsoft, and Philips, along with Compaq and Siemens Enterprise Network [7]. SALT is based on a collection of XML tags that can be embedded into an application written in other languages.

5.1 Goals

VoiceXML was originally designed to support telephone menus and other telephone functions [11]. It runs on voice browsers, normally located in data centers that are accessed through the public telephone network. SALT can also be used for telephone applications, but is designed for a more general purpose. The specification states that it should "enable multimodal and telephony-enabled access to information, applications, and Web services from PCs, telephones, tablet PCs, and wireless personal digital assistants (PDAs)" [8]. With multimodal access the user will be enable to interact in a variety of ways. Input with speech, keyboard, mouse; and output as synthesized speech, audio, plain text, motion video, and/or graphics.

5.2 Advantages and disadvantages

The primary advantage they both share is that they make the development of new voice applications much faster and less expensive. VoiceXML has the advantage that it is well known by interactive voice response (IVR) developers, who have lots of experience of voice applications. It allows them to deploy their voice application to a larger customer base. Additionally, the majority of the programmers understand HTML very well, and will also be able to work with SALT very easily.

Microsoft argues that SALT is easier to use because of its Visual Basic tie-in, whereas VoiceXML requires more telephony-type skills [7].

VoiceXML was designed as a standalone markup language and provides a full set of XML tags. SALT on the other hand only provides four primary tags that must be embedded into documents written in other languages [11].

SALT has a prompt-tag for speech output, a listen-tag for speech input, a DTMF-tag for telephone touch-tone input, and a smex-tag for call-control and messaging with other systems. Extensions have also been made to VoiceXML to support call-control.

One advantage that SALT has over VoiceXML is that it can use existing markup languages and web tools for developing new applications. VoiceXML requires special-purpose browsers and tools capable of handling the technology.

Which language is the best? Well it is up to the developer to choose. Some will standardize on VoiceXML, on SALT, or some combination of them both – depending of their applications and services. Right now, VoiceXML seems to have a bigger market than SALT, but when SALT is standardised, the battle between them will grow.

5.3 Code comparison

When comparing some code samples available at the VoiceXML Forum [9] we found that SALT applications seem to be more complex to build. SALT examples also appear to grow in size much quicker than the equivalent VoiceXML examples. This is because SALT authors have to write their own Form Interpretation Algorithm while VoiceXML authors get it for free. In appendix B we show a short example of the difference.

6 Our project example

6.1 Our vision

A service where a student could "look" at his/her schedule, book a group room, register for new courses, or sign up for an upcoming examination from a regular cell phone would be a very attractive service.

Imagine, you are running from the subway late for a lecture, as usual, and you don't know which room to go to. Instead of having to stop, open your bag and extract your schedule, you pick up your cellular phone and dial the service. Just say or enter your identification number and pin code, and your schedule will be read to you, so that you know where to go, all while on the run!

Or consider booking a group room for tomorrow, when you do not have any Internet access nearby for use by your computer. Pick up your cellular phone and dial the room booking service. You just say when and where you would like to book a room and the service checks if there are any rooms available at that time. If so, it books it for you and you are done.

6.2 VoiceXML hosting

With this in mind we have tried to create a small example to show how these services could be implemented using VoiceXML. To have VoiceXML working properly we also have had to implement a Text-To-Speech and Automated-Speech-Recognition engine. Since we thought that we did not have enough time to implement one of the free VoiceXML interpreters that exist on the Internet we chose another way.

There are many companies that offer free hosting of VoiceXML for developing. However, many of these do not offer a local access number and it would be extremely expensive to call from Sweden over the Atlantic all the way to Canada to test our application. However, VoxPilot [3], offers a local access number right here in Stockholm, so we chose to implement our application there. They have an online development environment to create and integrate VoiceXML applications and it is totally free to use. Like all other VoiceXML hosting companies they do not offer any support or designated phone numbers unless you buy their developer package.

6.3 Specification

After we decided that we would try our idea, we sat down and developed a specification, which resulted in a list of requirements:

- The service should be available in both English and Swedish.
- Each student should be authenticated with a 6-digit identification number and a 4-digit pin code.
- The database does not have to be secure, since we will not have time to implement this.
- Students should be able to check their schedule for the current week by specifying a day of the week. The server should return the course-code, time, and place.
- To book a group room the student should state a time and day. If a room is available it should be pre-booked by the server and the student has to confirm the booking.
- The server should hold information about all available examinations, including the time and place for them. Then the user could check on their exams by either specifying a course or picking one from a list to sign up for the exam.

6.4 Implementation

6.4.1 Database

We decided to build a small database to store all information about schedules, persons, and courses. For simplicity and to save time, we did not pay any attention to security of the database. We used Microsoft's Access database, since our hosting company did not support MS SQL or mySQL on their Windows accounts without paying an extra fee. Also, this database is well suited for small databases.

The database has a table, identity, which stores information about the user. The user's name is associated with an ID and his/her PIN code. The examination table stores information about available examinations that the user can sign up for. Each user is associated with different course codes together with the corresponding course name and date. The schedule table stores information about a user's schedule. For each day a text, the schedule that is read back to the user, is stored. The last table, grouprooms, stores information about the different rooms available at IT-Universitet in Kista, Sweden.

6.4.2 Application root document

Each document in our application is connected to the same application root document. The application root document stores information of selected language; the user's identity and name; if the user is logged in or not; and the different events that are thrown. An error event is thrown if the user is not logged in upon entering a document in the application. Another error event is thrown when the user has entered an invalid PIN code three times. In both these cases the user is disconnected. Our default handler that logs errors, outputs a message and disconnects, catches all other errors.

By having an application root document, all variables and events are reached by all other documents in the application as described in earlier chapters.

6.4.3 Grammars

The grammars in our application are small and simple to follow. They are written as internal grammars, direct in our VoiceXML document. Often we did not have to write any grammar at all, because they are “built-in” in the forms and menus. When the menus are output, the active grammars are the choices available. Many of the forms also have the attribute type set to digits, as the input only would consist of digits. As a result we do not have to write any grammars for these either.

6.4.4 The application

When a user calls up the service he/she will be prompted to choose a language: English or Swedish. The answer is then stored in the application root document. After the user has entered his/her personal identification number and PIN code a subdialog is started. This subdialog evaluates the information entered by the user. If the user entered the wrong login information an error will be thrown. Maximum retries have been set to three times. The identity of the user is stored in the application root document if the user passed the evaluation. Then he/she will be transitioned to the menu page. From here the user has three options: either to check his/her schedule, book a group room, or sign up for an examination.

6.4.4.1 Schedule

The schedule service is very simple. The user chooses a day to check their schedule. We thought that the service would probably be used when a student is late or does not have a computer or schedule on a paper available. Therefore the options available are simply today or tomorrow. After the selection a query is sent to the database that looks up the schedule based on who the user is and which day that was selected. A new document will be produced dynamically that reads the schedule back to the user.

6.4.4.2 Group rooms

A group room can only be booked the same day. The user says an approximate time and a query is sent to the database. Available rooms are returned, and if more than one is available they will be listed as alternatives to the user. If there are no group rooms available at that time, another time may be chosen by the user. He or she then has to confirm the reservation of the room to complete the booking. The room will then be booked in the name of this user. However, this feature is not completely implemented yet. We have encountered problems related to the alternative list of available rooms.

6.4.4.3 Examination registry

The database stores information about what examinations are available for each user. Examinations that are available for sign up are presented as a list one at the time. When entering the page a query is sent to the database that gathers the information based on the user’s identity. We have chosen to present the course code, the name of the course, and at what date the examination takes place. The user simply answers yes or no to sign up for the selected exam. When the list is empty the application returns to the main menu. This feature is not completely implemented yet.

6.4.4 Evaluation of our sample application

The total time we have spent on the implementation is approximately 20 to 25 hours. This includes a couple of hours of planning how we should do it. We looked at other examples to have a better understanding how the language worked.

During the implementation of our services we encountered some problems. We spent many hours searching on the Internet on how to create a dynamic VoiceXML document. When browsing a dynamic document in a regular browser such as Internet Explorer it worked. But when fetching it through the Voice Browser it failed. The error was that the <xml> tag, that specifies that it was an XML document, has to start at row one, position one. When we created our document through asp the <xml> tag was on the second row.

The biggest challenges were to understand the debugging information. The log viewer that VoxPilot supported was difficult to follow. It gave you information such as which document that generated the error and what type of error it was. However, it did not state where in the document the error occurred.

We have not been able to have the Text-To-Speech engine to interpret our dates correctly. Searching the Internet gave us many different ways to implement a date. We tried all of them but the output became a regular number. However, the log viewer stated that everything was correct and reported no errors. So this problem must reside in VoxPilot's Text-To-Speech engine.

7 Uses and business case

VoiceXML will probably be one of the most popular ways to develop voice applications for the telephone in the future. This new language is less complex and makes development cheaper than traditional ways of developing interactive voice response programs. Today, many Text-To-Speech engines do not feel natural to listen to. However, shorter phrases, numbers, and dates work just fine. Automated Speech Recognition engines have challenges to overcome, such as low bandwidth and a noisy environments. As soon as these engines become better the market will probably explode. Different booking services, such as hotel reservations, plane tickets, and hockey/football tickets, can be developed at a low cost by almost anyone.

8 Conclusion

When we have tried to build our own application we have found out a few things. VoiceXML is a great tool to create voice-enabled applications that are going to be used over the phone. The language is relatively easy to learn and to understand. With background knowledge of XML there is nothing more that the developer needs to know. It has both been fun and interesting to study a subject that has become so big on the market in the last few years. Many of the traditional voice services will probably be replaced in the near future.

The Text-To-Speech engines on the market vary in quality and we have a feeling that ScanSoft's [16] and Speechwork's [17] are the most used ones. TTS works pretty well when outputting short messages or numbers. But the best way to output a longer text is to use pre-recorded audio, because then it sounds more natural. However, the TTS engines are constantly under development and in a couple of years we might see Text-To-Speech engines that sound more natural, just like a human's voice.

The automated speech recognition already works very well. It has some drawbacks though; we tried our service while driving in a car, with the windows down and the radio on an average level (see section 4.2 Speech recognition in the telephony industry), and we found that it was hard, not to say impossible, to make the application understand what we said. However, it works in places without too much noise, and then it works very well. ASR engines will also be improved in the future and hopefully this problem has been solved by then.

References

Some of the references have been put directly in the text, often following a statement of some importance. Other references have been used to a lesser extent and just for general purposes like facts and check-ups, and are therefore not present in the report itself (but are of course in this list).

[1] XML-projektet om VoiceXML,
http://w3.msi.vxu.se/~sek/voiceXML/

[2] VoiceXML 2.0 reaches W3C milestone, John K. Waters,
http://www.adtmag.com/

[3] VoxPilot,
www.voxpilot.com

[4] Tellme Studio - An XML Primer,
http://studio.tellme.com/general/xmlprimer.html

[5] Voice eXtensible Markup Language (VoiceXML) Version 2.0,
http://www.w3.org/TR/2004/REC-voicexml20-20040316/

[6] VoiceXML lets you talk to computers, Network World, James Larson,
http://www.nwfusion.com

[7] New speech technologies making noise, Network World, Ann Bednarz,
http://www.nwfusion.com/news/2004/0322voice.html

[8] Speech Application Language Tags (SALT) Forum,
http://www.saltforum.org

[9] Voice eXtensible Markup Language (VXML) Forum,
http://www.voicexml.org

[10] An Introduction to Speech Recognition, Kimberlee A. Kemble,
http://www.voicexmlreview.org/Mar2001/features/recognition2.html

[11] Two Technologies Vie For Recognition In Speech Market, Neal Leavitt,
http://www.leavcom.com/pdf/Speech.pdf

[12] Computing Dictionary,
http://computing-dictionary.thefreedictionary.com/

[13] VoiceXML 2.0 Grammars, Part II, Jonathan Eisenzopf,
http://www.developer.com/voice/article.php/1565371

[14] The Fundamentals of Text-To-Speech Synthesis, Juergen Schroeter,
http://www.voicexmlreview.org/Mar2001/features/tts.html

[15] How Text-To-Speech Works,
http://project.uet.itgo.com/textto1.htm

[16] ScanSoft Incorporated,
http://www.scansoft.com

[17] SpeechWorks,
http://www.speechworks.com

[18] Using Mixed Initiative – Tellme Studio,
http://studio.tellme.com/vxml2/mixed/howto/mixed.html

[19] Open VXI VoiceXML Interpreter,
http://sourceforge.net/projects/openvxi/

[20] Is VoiceXML the right tool for you,
http://www.developer.com/voice/article.php/11062_1573371_1

[21] Dave Reggett's Introduction to VoiceXML,
http://www.w3.org/Voice/Guide/

[22] Speech Recognition Grammar Specification Version 1.0,
http://www.w3.org/TR/2003/PR-speech-grammar-20031218/

[23] VoiceXML: What's Everyone Talking About, Kevin Reichard,
http://networking.earthweb.com/netsp/article.php/10953_3349421_1

[24] Integrate VoIP into your enterprise infrastructure, Veronika Megler,
http://www-106.ibm.com/developerworks/wireless/library/wi-calvoip/

Appendix A, VoiceXML elements

<assign>	Assign a variable a value
<audio>	Play an audio clip within a prompt
<block>	A container of (non-interactive) executable code
<catch>	Catch an event
<choice>	Define a menu item
<clear>	Clear one or more form item variables
<disconnect>	Disconnect a session
<else>	Used in <if> elements
<elseif>	Used in <if> elements
<enumerate>	Shorthand for enumerating the choices in a menu
<error>	Catch an error event
<exit>	Exit a session
<field>	Declares an input field in a form
<filled>	An action executed when fields are filled
<form>	A dialog for presenting information and collecting data
<goto>	Go to another dialog in the same or different document
<grammar>	Specify a speech recognition or DTMF grammar
<help>	Catch a help even
<if>	Simple conditional logic
<initial>	Declares initial logic upon entry into a (mixed initiative) form
<link>	Specify a transition common to all dialogs in the link's scope
<log>	Generate a debug message
<menu>	A dialog for choosing amongst alternative destinations
<meta>	Define a metadata item as a name/value pair
<metadata>	Define metadata information using a metadata schema
<noinput>	Catch a noinput event
<nomatch>	Catch a nomatch event
<object>	Interact with a custom extension
<option>	Specify an option in a <field>
<param>	Parameter in <object> or <subdialog>
<prompt>	Queue speech synthesis and audio output to the user
<propterty>	Control implementation platform settings
<record>	Record an audio sample
<reprompt>	Play a field prompt when a field is re-visited after an event
<return>	Return from a subdialog
<script>	Specify a block of ECMAScript client-side scripting logic
<subdialog>	Invoke another dialog as a subdialog of the current one
<submit>	Submit values to a document server
<throw>	Throw an event
<transfer>	Transfer the caller to another destination
<value>	Insert the value of an expression in a prompt
<var>	Declare a variable
<vxml>	Top-level element in each VoiceXML document

Table 4, VoiceXML elements, [5]

Appendix B, VoiceXML vs. SALT

Transitioning to another page:

```
<html xmlns:salt="http://www.saltforum.org/2002/SALT" >
   <body>
      <form id="form1" action="nextpage.html">
         <input type="button" onclick="transition();" value="next page" />
      </form>
      <salt:prompt id="transitionPrompt">
         Let's go to the next page!
      </salt:prompt>
      <script>
         function transition()
         {
         transitionPrompt.Queue();
         PromptQueue.Start();
         form1.submit();
         }
      </script>
   </body>
</html>
```

Figure 4, Transition.salt, (source: http://www.voicexml.org/salt/comparisons.html)

```
<?xml version="1.0"?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/voice">
   <form>
      <block>
         Let's go to the next page!
         <goto next="nextpage.vxml"/>
      </block>
   </form>
</vxml>
```

Figure 5, Transition.vxml, (source: http://www.voicexml.org/salt/comparisons.html)