Can Web understand Croatian language?

Albert Novak
Croatian Academic and Research Network
Albert.Novak@CARNet.hr

Abstract: Today we have new standards from the W3 consortium to access the Web-based services. The special browser called "Voice Browser" allows any telephone to be used to access the appropriately designed Web-based services and can be of help to people with visual impairment. It can also allow effective interaction with the Web content in the cases where using the mouse and keyboard may be inconvenient. It is the intention of this paper to explain how we can use VoiceXML to access from "Voice Browser" to Web-based services and how can we appropriately build the Croatian language speech recognition system as part of the "Voice Browser".

1 Introduction

Nowadays we are witnesses of new technologies and services, which are constantly changing toward better improvement of communication capabilities between users and services. Some traditional ways of communication such as keyboard and display do not always prove to be the easiest and most practical ways. In some situations where it is impractical to use keyboard, the natural speech turns out to be the easiest way for data inputting.

We can imagine these two situations; in first one we are using the only old telephone system for communication with our web application, and in second one we are combining the traditional way with speech capability. In both situations we must have a language, which describes how we can incorporate speech capabilities into the web application. There are two standards; VoiceXML, and SALT (Speech Application Language Tags)[3][11]. For each language there is an interpreter which understands language and can connect our client application with web. When we are using voice-only application with old telephone system, the interpreter is called "Voice browser"[2], and when we are combining GUI (Graphical User Interface) with voice, the interpreter is called "Multi-modal browser"[8].

2 VoiceXML

VoiceXML is an XML language for writing Web pages you interact with by listening to spoken prompts and jingles, and control by means of spoken input[1]. VoiceXML is different from HTML. Speech-based interface to web application must have control over user-application interactions. HTML is designed for visual Web pages and with HTML we are not having control over communication between user and application. On the other hand, VoiceXML is designed to give us control over spoken dialog between the user and the application. During dialog between user and application each one takes its turn to speak. In each turn, application first prompts user, and user in his turn responds or uses DTMF tones (touch tones) on his telephone.

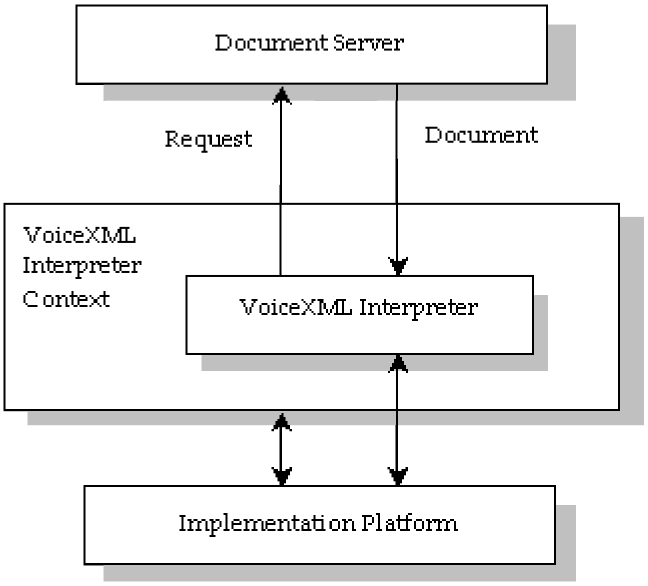


Figure 1: VoiceXML Architecture [3]

The architectural model of VoiceXML is given in figure 1. When user dials VoiceXML application, VoiceXML Interpreter requests root application document from a web server (document server). Then the Web server replays with a dynamically generated or static document. VoiceXML Interpreter processes received document and based on that control conversation between user and application. VoiceXML Interpreter and VoiceXML Interpreter Context monitor in parallel user inputs through Implementation Platform. In this way, VoiceXML Interpreter Context may always monitor for special phrases that can execute some general activities such as connections with human operator or can change characteristics of text-to-speech engine.

The Implementation Platform is responsible for communication with external world, and is controlled by the VoiceXML interpreter context and by the VoiceXML interpreter. The VoiceXML 2.0 specification defines that it must support: audio output using audio files and text-to-speech (TTS), detecting and reporting character (DTMF) and/or spoken input simultaneously and to control input detection interval duration with a timer whose length is specified by a VoiceXML document, and making a third party connection through a communications network, such as the telephone[3]. In real voice interactive application, VoiceXML Interpreter Context is responsible for the first contact with user: waiting for user incoming call, answering on that call, acquiring the initial VoiceXML documents. VoiceXML Interpreter coducts dialog between user and application, and Implemention Platform generates events in accordance with user action. VoiceXML Interpreter processes events specified by VoiceXML document while other events are processed by VoiceXML Interpreter Context.

2.1 Key Concepts

A session is a process that starts with a telephone call and lasts until the communication is disconnected. During the session, dialogs that the user is using to communicate with application are turning. The session always ends after the disconnection of telephone call and it can be caused by user, VoiceXML document or Interpreter context request (Implementation Platform disconnects call).

In VoiceXML concepts, VoiceXML documents define an application through the set of dialogs between user and system. The application is proceeding through the sequence of dialogs that are alternating in accordance with VoiceXML documents. The user is always in a certain dialog and at the end of one dialog he transits to another dialog or ends the session.

Documents that describe application share the same context defined by application root document. The root document is automatically loaded whenever one of application documents is loaded and stays active until any of application documents is active (Figure 2). The application root document will be unloaded only if there is a change of application (loads document from another application) or at the end of session.

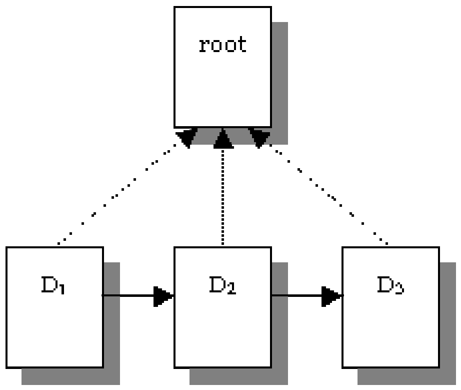


Figure 2: Root document

The basic element of VoiceXML application is dialog. It consists of menus and forms. Forms collect data from user defined by form's input fields and menus request from user to choose one of the offered options. Inside the forms many fields that collect different data from user through the dialog can be created. Every filed has specified grammar that defines which input data it can receive. Similar to the ordinary HTML applications data collected by form can be send to the web server for further processing. Menus are offering users several options and after the selection of some option application can change dialog or even application.

Similar to program languages, VoiceXML enables using of variables and functions. Variables can be defined in any level and their scope follows an inheritance model. A Subdialog has similar behaviour as a function call. During the dialog we can call subdialog and make a new conversation inside it. At the end of conversation we return to the original dialog. All parameters before the subdialog call will be saved and after return from subdialog will be restored. Subdialog can be used to create library of common tasks.

Implementation platform will generate event if user doesn't respond to a prompt (respond timeout) or when the speech recognition system doesn't recognize user's respond. VoiceXML allows handler writing that can take over event processing. Handler follows inheritance model and if there isn’t adequate handler at dialog level it will be taken over by handler at a higher level and etc.

VoiceXML allows additional control over the application using the script language – ECMAScript (JavaScript).

2.2 VoiceXML examples

Here is a very simple VoiceXML application similar to the first "Hello World" program. It says "Dobrodošli na CARNetove službene stranice", plays a short audio advertise jingle. After that synthesis engine announces news "Poslušajte CARNetove najnovije vijesti", plays prerecorded news, and then exits. Prompts can consist of any combination of prerecorded files, audio streams, or synthesized speech.

```
<?xml version="1.0" encoding="iso-8859-2"?>
<vxml version="2.0" lang="hr">
<form>
<block>
<prompt bargein="false">Dobrodošli na CARNetove službene stranice!
<audio src="http://www.CARNet.hr/welcome.wav"/>
Poslušajte CARNetove najnovije vijesti
<audio src="rtsp://www.CARNet.hr/news.wav"/>
</prompt>
</block>
</form>
</vxml>
```

In the next example we are offering a menu to the user and he can choose type of news: users, network, education, and administrators.

```
<?xml version="1.0" encoding="iso-8859-2"?>
<vxml version="2.0" lang="hr">
<menu>
  <prompt>
    Odaberite područje vijesti: <enumerate/>
  </prompt>
  <choice next="http://www.carnet.hr/users.vxml">
    CARNet korisnici
  </choice>
  <choice next="http://www.CARNet.hr/network.vxml">
    mreža
  </choice>
  <choice next="http://www.carnet.hr/educa.vxml">
    obrazovanje
  </choice>
  <choice next="http://www.carnet.hr/admin.vxml">
    sistemci
  </choice>
</menu>
</vxml>
```

We can imagine a dialog following the previous example between user and server:

Computer:	Odaberite područje vijesti: CARNet korisnici; mreža; obrazovanje; sistemci.
User:	Sigurnost
Computer:	Nisam razumio što ste rekli. (a platform-specific default message)

Computer:	Odaberite područje vijesti: CARNet korisnici; mreža; obrazovanje; sistemci.
User:	CARNet korisnici
Computer:	(proceeds to http://www.CARNet.hr/users.vxml)

How can we use a form for collecting data, you can see in the next example. Server, using a form, asks the user to choose a city and the number of participants for videoconference.

```
<?xml version="1.0" encoding="iso-8859-2"?>
<vxml version="2.0" lang="hr">
<form>
<field name="city">
<prompt>S kojim gradom želite održati videokonferenciju?</prompt>
<option>Dubrovnik</option>
<option>Osijek</option>
<option>Pula</option>
<option>Rijeka</option>
<option>Split</option>
</field>
<field name="participants">
<prompt>Odabrana je <value expr="city"/>Koliki je predviđeni broj
učesnika?</prompt>
</field>
<block>
<submit next="http://www.CARNet.hr/videoconf_handler" namelist="city
participants"/>
</block>
</form>
</vxml>
```

3 SALT – Speech Application Language Tags

Speech Application Language Tags or SALT is a small set of XML elements and their associated attributes, events and methods that add speech and telephony call-control features to existing Web-based application. SALT, different from VoiceXML, isn't independent language and its elements are embedded in existing HTML, XHTML an XML pages [12].

SALT and VoiceXML have different technical goals. Whereas VoiceXML is designed for the development of telephony-based application, SALT focuses on adding multimodal capability to web-based application. Multimodal applications have a capability to improve classic web-based applications with speech and telephone call-control features. Multimodal applications enable users to interact with it using combination of traditional ways such as a keyboard, keypad, mouse and new approaches like speech or DTMF. Similar is with presentation of output data because we can combine output on classical display with speech. In Multimodal browser it is possible to use independently or concurrently both approaches.

VoiceXML and SALT have different realisation of applications. VoiceXML uses a document-based approach, where applications are built by one or more documents that have tags which describe dialogs. On the other side, SALT takes programming approach, where applications are made from objects, triggers and events. SALT uses intensively JavaScript to build the application because speech recognition engine generates event after user's respond and through JavaScript handler SALT processes this respond.

SALT architecture is similar to the VoiceXML architecture. Both architectures have on one side document server and on the other side client application. SALT client application is voice or multimodal browser capable to be used on whole continuum of devices from PCs to mobile devices. SALT voice browser, like VoiceXML voice browser, is exclusively used from telephone through POTS (Plain Old Telephone System) or VoIP (Voice over IP) and there is no functional difference between SALT and VoiceXML. We can see SALT architecture on Figure 3.

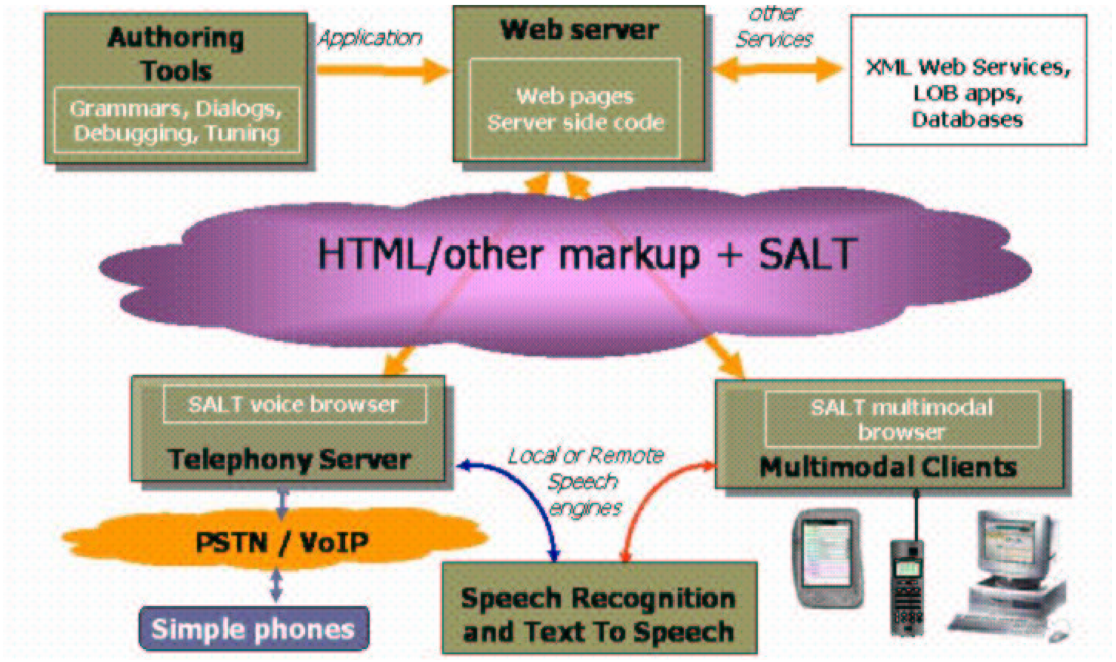


Figure 3: SALT Architecture [10]

3.1 SALT elements trough examples

SALT has very small set of tags. There are three main top-level tags:

<listen …>	configures the speech recognition engine, executes recognition process and handles speech input events
<prompt …>	configures the text-to-speech engine and plays out defined prompts
<dtmf …>	configures and control DTMF in telephony applications

The <dtmf> and <listen> top-level tags contain elements <grammar> (defines used grammar) and <bind> (connects input filed with user respond). The <listen> element can also contain <record> element (records user respond).

SALT elements are XML objects in the Document Object Model (DOM) of the pages. As any programming objects, SALT elements contain methods (object functions), properties (object variables), and event handlers (functions which processing generated events), which are accessible to the script. It can also interact with other processes and events in execution of the web pages. Those properties allow SALT's to be seamlessly integrated into existing web applications.

In our first example we use famous "Hello World" program, or more precisely SALT application, which says "Hallo World".

```
<html xmlns:salt=http://www.saltforum.org/2002/SALT>
  <body onload="hello.Start()">
    <salt:prompt id="hello">Hello World</salt:prompt>
  </body>
</html>
```

In this very simple example we can see how simple it is to add SALT tag into a HTML application. When this example is loaded in SALT compatible browser (for this example it is not important if that voice or multimodal SALT browser) it initiates text-to-speech (TTS) engine trough the method "Start()" and speech synthesiser generates waveform which sounds like "Hallo World".

Next, more complicated, example shows us how can we use SALT speech recognition capabilities. This is simple application through which we can get basic information about CARNet projects. When the application is loaded in SALT browser control is given to JavaScript function "RunApp()" which controls a conversation. At the beginning the application says welcome message (using "Welcome.Start()" function) and asks user to say the name of the project. Function "recoProjectName()" initiates speech recognition using external XML grammar. When speech recognition engine stops with recognition it generates event "onreco" and handler function "processProjectName()" takes control. That function binds recognized user respond with input field "ProjectName" and submits form data to the server-side script "projectinfo.php".

```
<html xmlns:salt="http://www.saltforum.org/2002/SALT">
  <body onload="RunApp()">
    <form id="ProjectForm" method="post" action="projectinfo.php">
      <input name="ProjectName" type="text"/>
    </form>
    <salt:prompt id="Welcom">
      Dobrodošli na CARNetov katalog projekata
    </salt:prompt>
    <salt:prompt id="AskProjectName">
      Molim vas ime traženog projekta
    </salt:prompt>
    <salt:listen id="recoProjectName" onreco="processProjectName()">
      <salt:grammar src="projects.xml"/>
    </salt:listen>
    <script>
       function RunApp()
       {
         if(ProjectForm.ProjectName.value=="")
         {
           Welcom.Start();
           AskProjectName.Start();
           RecoProjectName.Start();
         }
      }
      function processProjectName()
      {
        ProjectForm.ProjectName = recoProjectName.text;
        ProjectForm.submit();
      }
    </script>
  </body>
</html>
```

4 Voice and multimodal browser

Different from standard browser, Voice and Multimodal browsers have the capability to improve web applications with speech. In the case of Voice browser, the only communication channel between client and Voice browser or Voice server is POTS or VoIP. Voice browser is designed only for speech communication and the client is always some kind of telephone equipment (classical or VoIP telephone).

On the other side, Mutlimodal browser has a capability to be used in parallel classical graphical user interface (keyboard, mouse, display, etc.) and speech interface with DTMF capability.

With VoiceXML standard we can build only Voice browser that can be used only for telephone applications. Around SALT standard we can build both type of browsers. VoiceXML is an older standard and momentarily is actual VoiceXML version 2.0. Several vendors have implemented VoiceXML 1.0 and some of vendors follow new working draft VoiceXML 2.0. SALT is new standard and today we don't have many implementations.

Unfortunately, commercial solutions are very expensive and it is very difficult to find speech implementation with Croatian support. If we want to build voice or multimodal applications we must first build adequate browser with Croatian support. Carnegie Mellon University has developed open source implementation of VoiceXML interpreter and is currently developing SALT interpreter. OpenVXI is an open source implementation of VoiceXML interpreter and OpenSALT is an open source implementation of SALT interpreter.

4.1 OpenVXI

OpenVXI isn't a complete solution for VoiceXML browser; it is only one component of a complete VoiceXML platform. Figure 3 shows the components of a VocieXML system. The speech browser platform has four parts[14]:

1. **An operation administration and maintenance (OA&M) system and main process**. This collection of tools is responsible for system management and error reporting. Also it invokes the speech browser within a thread to begin execution.

2. **The OpenVXI**. It interprets VocieXML markup language and invokes implementation platform (speech recognition, text-to-speech engine, telephone call control) to render the markup.

3. **The Platform Components**. The platform components provide the services necessary for the system to run. The necessary components are recognition engine, prompt engine, Internet fetch library and ECMAScript (JavaScript) engine. The OpenVXI interacts with those components through the interfaces (API) that must be implemented for the system to run properly. The mechanism for communications between interfaces implementations and components engines isn't defined: for communication we can use direct approach or client/server strategy.

4. **The Telephony and base services**. These services are necessary to receive phone calls. Also, these services must have adequate hardware support capable to control phone calls (receive, transfer, disconnect, and wait for calls).

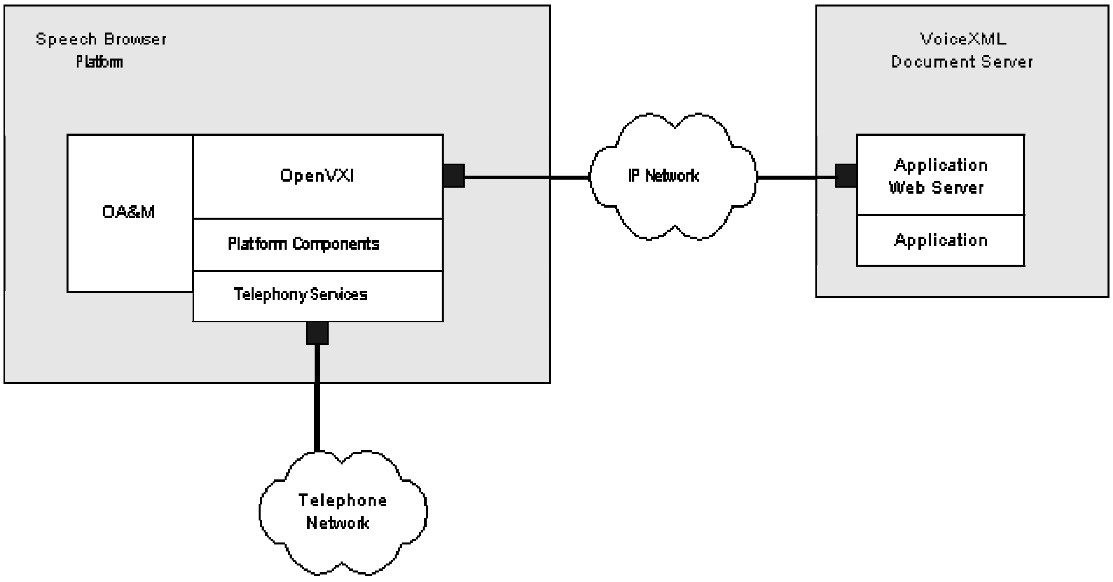


Figure 4: OpenVXI System Architecture [13]

The OpenVXI toolkit doesn't have components necessary for full functional Voice browser. If we want to build complete VoiceXML platform we must to include missing parts.

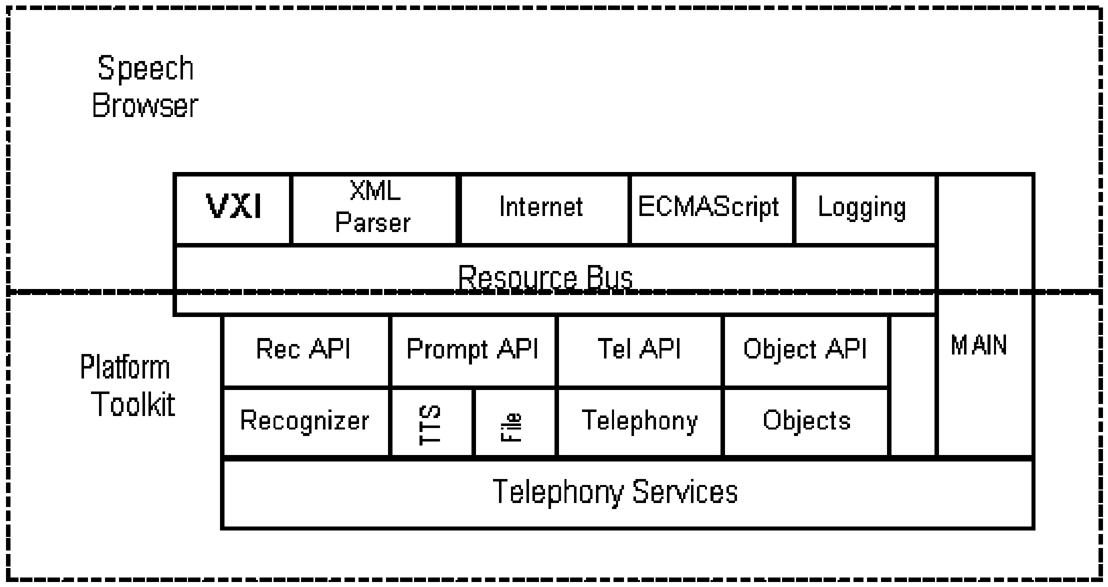


Figure 5: OpenVXI Toolkit Architecture

Figure 5 shows OpenVXI toolkit architecture with all components. We have two main parts: speech browser and platform toolkit. The Speech Browser parts are[14]:

1. **VXI**. This component interprets all VoiceXML markup and controls main loop. OpenVXI toolkit implements fully VoiceXML 1.0 language and follows VoiceXML 2.0 (we don't have support for inline W3C grammars within CDATA tags).

2. **XML Parser API**. It provides access to XML DOM parser and currently is implemented by directly calling the open source Apache Xerecs and DOM parser API.

3. **ECMAScript (JavaScript) API**. It provides access to the JavaScript services, currently by integrated open source Mozzila SpiderMonky JavaScript interpreter.

4. **An Internet and cache Library API**. This component provides access to application documents via http or ftp protocol and has support for POST method of data sending. OpenVXI toolkit integrates open source W3C Libwww library.

5. **A Logging interface**. It is used to report events, diagnostics and error messages to the system. OpenVXI toolkit implements only basic logs to the file and optionally to the standard output.

Platform components of OpenVXI toolkit are only implemented as simulators. If we want to build the system completely we have to integrate it with real speech and prompt engine. For example, instead of the speech and prompt simulator we can implement interfaces to the open source speech recognition engine sphinx and open source text-to-speech engine festival.

4.2 OpenSALT

Unfortunately, SALT is new standard and there is no open source implementation available yet. Carnegie Mellon University is starting an open source project called OpenSALT. The project will produced an open source SALT 1.0 compatible browser. The first release will be completed at the end of 2002. The browser base is open source Mozilla web browser, Spinx recognition and Festival synthesis software. Open SALT browser is going to support Windows and Linux platform, but Windows release is going out first and Linux version is expected later.

5 Speech Recognition and Synthesis

The previous chapters have explained basics of standards and products important for building a completely functional web-based speech system. If we want to interact with web using speech we have to use some of speech recognition engine embedded in our system. Also, if we want to prompt user with computer synthesised speech we must embed text-to-speech engine to our system.

Today, we can find on the market many commercial products and also many commercial complete solutions for Voice browsers. But, those products are expensive and usually don't have support for exotic languages. In some situations the only solution is to build a new system with available components. For most of Academic community those are open source solutions or products. Some of those solutions are Sphinx for speech recognition and Festival for text-to-speech synthesis.

5.1 Sphinx

The Sphinx Group from Carnegie Mellon University has released a group of products for building speech recognition engine. The most important products are: sphinx 2 and sphinx 3 for speech recognition, SphinxTrain for acoustic models training, web-based Language Modelling Tool and CMU-Cambridge Statistical Language Modelling Toolkit for building language models[16].

Both speech recognition engines use 5-state HMM (Hidden Markov's Models) for phones describing. Sphinx 2 is a real-time, large vocabulary, speaker independent speech recognition engine, distributed as free software under Apache-style license. Sphinx 3 is a slower but more accurate speech recognition engine. Sphinx 2 is "semicontinuous" (uses tied mixtures), and Sphinx 3 uses fully continuous observation densities (untied, so that each state has its own distribution statistics). Sphinx 2 is portable and can be used on various types of devices that require short response time. Sphinx 3 can be used for slower (not real time) but more accurate applications like broadcast news transcriptions or acoustic trainer.

If we want to build a successful speech recognition system we must have appropriately trained acoustic models. Acoustic models describe how the basic sound units (usually phone but sometimes can be word) change across time. Each phoneme or word is modelled by a sequence of states and signal observation probabilities called HMM. SphinxTrain is a tool for building acoustic models for CMU Sphinx 2 and Sphinx 3 engines. With it we can train acoustic models for any language, task or communication channel conditions that will allow the run-time engines to recognize trained speech. Acoustic models training requires enough data, effort and pure hardware resources to build accurate and task adapted acoustic models.

Speech engines use language models to improve accuracy of speech recognition system. The language models describe likelihood of appearing word sequence. Usually a N-gram model is used where N describes the number of involved word. For example, Sphinx 2 uses trigram (3-gram) models where sequences of three words are observed. We have two tools to build language models: web-based Language Modelling Tool and CMU-Cambridge Statistical Language Modelling Tool. With the first tool we simply upload a set of sentences and web-engine creates language models for us. The second tool is more elaborate and powerful.

Language models, plus acoustic models, plus Sphinx engine give us everything we need to build a speech recognition system.

5.2 Festival

Text-to-speech engine is another important part of Voice browser platform components. With it our Voice browser is capable to pronounce specially markup part of VoiceXML documents. Festival is a complete open source solution for building text-to-speech system [17]. With Festival we can build new voices and play existing voices. Within the Festival we can identify three parts of the TTS process:

1. **Text analysis.** To identify words and utterances from raw text.

2. **Linguistic analysis**. Finding pronunciations of the words and assigning prosodic structure to them: phrasing, intonation, and durations.

3. **Waveform generation**. To generate waveform from a fully specified form (pronunciation and prosody).

The process of creating new voices is not trivial, but a dedicated person with experience in speech recognition, computational linguistic and/or programming can probably build a new voice in a week[17]. Unfortunately, the process of building new voices is very far from automatic generating of new voices. The good behaviour of Festival is the possibility to use external processes to perform waveform synthesis. Festival can use text and linguistic analysis from MBROLA project to generate waveform. All language already supported with MBROLA can be used with Festival, and Croatian is one of MBROLA languages.

6 How can we build the Croatian Speech Recognition System?

The Voice or Multimodal browser can be build using OpenVXI or OpenSALT toolkit, Sphinx as speech recognition engine, and Festival as text-to-speech engine. Both engines must understand Croatian language. With Festival we have an easier job because Festival can use text and linguistic analysis from MBROLA project to generate waveform. MBROLA project supports Croatian language and thus we can use results from MBROLA project with Festival engine to get Croatian text-to-speech engine.

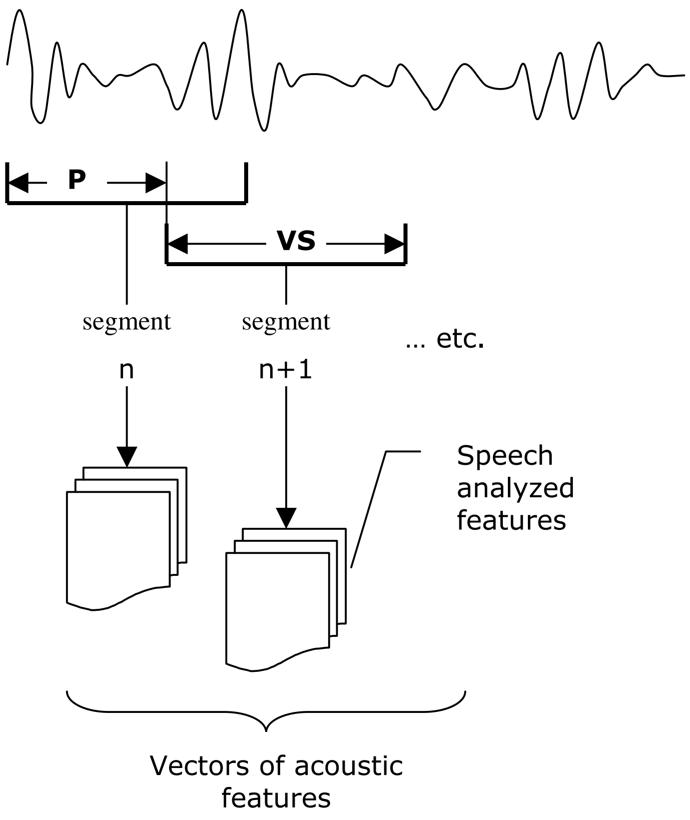
Now, we have a problem only with speech recognition engine, or more precisely, with Croatian acoustic and language models for Sphinx engine.

Carnegie Mellon University has built a system for real-time Croatian to English and reverse speech translation. For that purpose they have built Croatian acoustic and language models but those models aren't available to public. We only have the solution to build our acoustic and language models. This is not a big problem because we have tools from Sphinx group for acoustic and language building. Acoustic and language models built for our dedicated application can be more accurate from general models and our web application will be more successful.

6.1 Speech Recognition Theory

Speech recognition system transforms input acoustic signals in appropriate set of phonemes or words in the process called “Speech recognition”. For that purpose the digitised speech signal is first transformed into a set of useful representatives of signals or features at a fixed rate, typically once every 10 to 30 msec (). These features are then used to search for the most likely word candidate, determined by the acoustic, lexical, and language models. Throughout this process, training data are used to determine the values of the model parameters.

Figure 6: Features extraction



A vector o_t of acoustic features is computed every 10 to 30 msec (P) from digitalised speech frame VS (25~30msec). We use a set of transformations in features extraction method to convert acoustic signal into few representatives of them. In that process we try to reduce redundancy from acoustic signal and decrease the influence of noise.

Sequences of parameter vectors are assumed to form an exact representation of speech waveform and can be treated as observations of acoustic word models used to compute $p(o_1^T \mid W)$, the probability of observing a sequence o_1^T of vectors when a word sequence W is pronounced. Given a sequence o_1^T, a word sequence $\widehat{W}$ is generated by the speech recognition system with a search process based on the rule [20][22]:

$$\widehat{W} = \arg\max_{\mathrm{W}} p(o_1^T \mid W) p(W)$$

$\widehat{W}$ is candidate with a maximum a-posteriori probability (MAP). Probability $p(o_1^T \mid W)$ is computed by Acoustic Models (AM), while probability $p(W)$ is computed by Language Models (LM).

6.2 HMM and Acoustic models

The most used recognition paradigm in the past twenty years is known as Hidden Markov Models (HMM). The HMM is a doubly stochastic model, in which the generation of the recognized phoneme or word string and the frame-by-frame features extracted from acoustic signal are both represented probabilistically as Markov processes [21].

A hidden Markov model is defined as a pair of stochastic process (X, O). One of the processes, X is a first order Markov chain and isn't directly observable (is hidden). Second process O is sequence of random variables taking values in the space of acoustic parameters, and is observable.

If $o \in O$ is acoustic vector of observed speech signal, and $i, j \in X$ is a state of HMM, then HMM model is defined with:

$A \equiv \{a_{i,j} \mid i, j \in X\}$ transition probabilities

$B \equiv \{b_j(o) \mid j \in X\}$ output distributions

$\Pi \equiv \{\pi_i \mid i \in X\}$ initial probabilities

where is:

$a_{i,j} = p(X_t = j, X_{t-1} = i)$ probability of model to be in state j at the time t if previous state is i

$b_j(o) = p(Y_t = o \mid X_t = j)$ probability of model to generate vector or symbol o in the state j at the time t

$\pi_i = p(X_0 = i)$ probability of model to be in the state i at the time t (on the beginning).

We have three problems when we use HMM for speech recognition:

Evaluation – How do we compute probability $p(O_1^T \mid \Theta, M)$ of HMM model M with parameter $\Theta = \{A, B, \Pi\}$ generating an output sequence O_1^T?

Decoding – How do we compute the most probable state sequence X_1^N of model M with parameters $\Theta = \{A, B, \Pi\}$ if we have an output sequence O_1^T?

Training – How do we adjust the model parameters $\Theta = \{A, B, \Pi\}$ to maximize likelihood of the model M generating the output sequence O_1^T?

Solutions of these problems give us powerful tool for speech recognition (more about recognition and training algorithms can be found in [18],[19],[20] and [22]).

We can classify HMM according to the nature of distribution function of matrix B on:

1. Discrete HMM
2. Semicontinuous HMM
3. Continuous HMM

The simplest solution is discrete function of output distribution. In each frame or observed segment of speech signal we get one symbol in a finite alphabet of **N** elements that is representatives of that segment. Technically speaking the vector quantizer (VQ) divides vector space into N regions and every vector is replaced by one symbol (numerical identifier) from codebook. Each state of HMM has defined output distribution as a histogram (with N bins) of the occurrence-frequency of each symbol. Discrete HMM assumes that all data within a region is equally probable. Performance of discrete HMM is strongly dependent upon accuracy of vector quantizer.

Continuous HMM has completely different assuming. Output distribution of each state is represented by mixture density that defines the part of acoustic space where vectors occur. Usually we use Gaussians distribution, and each mixture component has a mean and variance. The boundaries between regions aren't clear and partitions aren't rigid. HMM probability to generate observed vector in each state is calculated as a sum of probability to generate observed vector of each mixture components in that state. Continuous HMM must have very large training database.

The third approach is between discrete and continuous models. Semicontinuous or tied mixture HMM represents all vectors by continuous density codebook. Semicontinuous model doesn't have different distribution for each state of model, but all states of all models share common mixture density. The output distribution of state is a sum of mixture of continuous density from codebook normalized with "mixture weight". The entire acoustic space is covered by a set of independent, usually Gaussian, densities. Those densities are obtained in the similar way as VQ codebook in discrete HMM, and are represented by means and covariance stored in a codebook. Semicontinuous HMM has a much lower computational complexity than the continuous model and it can model acoustic space much more accurately than the discrete model.

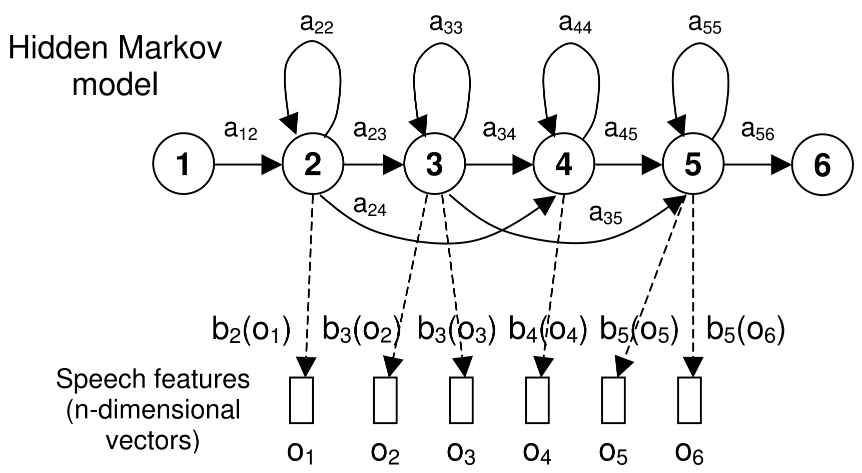


Figure 7: Hidden Markov Model with six states

In the modern speech recognition system, the acoustic model describes the basic unit of acoustic signals, usually phoneme or word, with n-state hidden Markov model (Figure 7). Models, usually, have five to seven states where the first and last states are transitional states. With transitional state we can connect two models in a more complex representatives of speech and thus build continuous speech recognition system.

6.3 Language Model

Language model gives context or recognition environment for speech recognition system. Every language has rules that describe relation between words in sentences. Language model has embedded knowledge about relation between words of language and describes the likelihood, probability, or penalty taken when a sequence or collection of words is seen. The probability $p(W)$ of word's sequence $W = w_1,...,w_n$ can be computed by Language model using following expression:

$$p(W) = p(w_1, w_2,..., w_n) = \prod_{i=1}^{n} p(w_i \mid w_0,..., w_{i-1})$$

Speech recognition system, usually, use N-gram model for language modelling. Usually has been used trigram model where is accumulated knowledge about sequence of three words [23].

Language model can significantly improve accuracy of speech recognition system. Language model tuned for particular application gives the best results (this is specially true for small vocabularies).

6.4 Acoustic and language modelling

Training of speech recognition system is difficult task. Through the training process, acoustic and language models must be trained to increase accurancy of speech recognition system. These two models are independent and can be trained separately.

Acoustic models are representing acoustic characteristics of base recognition unit. A basic unit can be a phone or a word. A phone will be chosen when we build large vocabulary speech recognition system. Word basic unit would be a better solution if we built a small vocabulary (50~60 words) system.

Relatively slow articulators produce speech waveforms (articulators are human organ responsible for speech waveforms generating) and a particular phone depends on a preceding and following phone or simply context. We call this phenomenon coarticulation. Coarticualtion effect can be modelled through context-depended phone models. The amount of context is defined by chosen model:

Monophone. This is the simplest solution in which surrounding context is not used.

Allophone. This model has not got a surrounding context, but uses multiple representatives of phone.

Biphone. Each phone model is represented only with a left or right context.

Triphone. Each phone model is represented by both a left and right context.

After we have chosen phone model, in the next step we must define the used phones and create a dictionary. The dictionary describes pronunciation of all words from training database, and same word can have different pronunciation. Pronunciation is described by chosen phones. The dictionary also describes some non-speech sounds like silence, background noise, cough, etc. These sounds must be defined on phone's list as basic models.

The training database must be closely connected with real application. That means if we are building telephone application, it is desirable to create training database from recorded telephone speech. Also, the training database must have all expected words of future application. If we would like to build speaker independent speech recognition system, the training database must have a different speaker recorded (different age, different gender, etc). The training database must have a transcription of the recorded speech.

The last decision is a type of HMM. The most important question is the number of state and type of output distribution. The number of state of HMM is in relation with the phone model (or word model). If we would like to train acoustic model based on word, usually 6 to 7 states are used, and in the case of triphone models 3 to 5 states are used. Choice of output distribution can be semicontinuous or continuous and depends on training database. Continuous acoustic models are more accurate but must have bigger training database for appropriate training.

Language modelling is important to reduce complexity of speech recognition task and increase recognition accuracy. In a speech recognition system, usually a statistical N-gram model is used. A powerful behaviour of N-gram modelling is the fact that a model can be directly estimated from text data (model dose not depend on the acoustics). For trigram estimate can be described as occurring-frequency of some trigram normalized with sum of occurring-frequency of all trigram which have the same first two words:

$$p(w_k \mid w_i, w_j) \approx \frac{c(w_i, w_j, w_k)}{\sum_{w_l} c(w_i, w_j, w_l)}$$

This simple approach has the "unseen N-grams" that don't occur in the training set will be given a zero probability, and that N-grams cannot be recognized. One solution for that problem is to hold some amount of probability to unseen N-grams (back-off method). In the case of trigram, unseen trigram will be estimated using bigram on reserved probability mass. The same process happens with bigrams, when unseen bigram will be estimated using unigram on reserved probability mass of unseen bigram [23].

Training database for Language modelling is a corpus of text data adequate for our application (e.g. we mustn't use medical corpus for language modelling to flay reservation system). We must only choose ratio discount for back-off calculation. Usually this is 0.5 small corpora and will be applied to all counts (that means half of all probability mass will be reserved for unseen N-grams, and half of reserved probability mass will reserved for unseen N_{-1}-grams). Small corpora means < 50k words, and for larger corpora ratio discount can be smaller because we must reserve smaller probability mass for unseen N-grams.

Acoustic and language models are all we need to build speech recognition system. Tools for acoustic and language mentioned earlier can be found on Internet [16].

7 Conclusion

Can the web understand Croatian language? Yes, it can! We have standards that define markup language necessary for speech communication with web application. These standards enable us to build web application that can understand Croatian language. But this is not a simple task.

In the simplest solution we can develop application from prerecorded Croatian sentence. In that case we don't need text-to-speech engine. Users responds can be input using DTMF (touch tone). Maintenance of this system is complicated because for every change in application dialog we must prerecord new sentences, and in that case we could use text-to-speech engine such as Festival. Festival can be integrated with existing open source toolkit (OpenVXI and OpenSALT) to build a complete solution possible to generate Croatian speech waveforms.

The most complicated solution can be used to develop complete Voice or Multimodal browser for Croatian language, which is possible to understand and talk Croatian. That can be done with OpenVXI or OpenSALT (VoiceXML or SALT interpreter), Sphinx (speech recognition engine), and Festival (text-to-speech engine). Integration of these components demands some programmer's knowledge but it is not an impossible task. Integrated system is only one part of the problem and acoustic and language modelling is another part. For modelling we have to collect and prepare adequate training database and that job request time and resource. At the end we will get a very usable system and investment will return to us in simplified development of further voice web based applications.

We have the technology and if we use it in the right way we can achieve the aim, "build a useful Croatian voice portal", with more or less efforts.

8 References

[1] David Raggett, "Getting started with VoiceXML 2.0", http://www.w3.org/Voice/Guide/

[2] W3C, "Woice Browser" Activity – Voice enabling the Web!, http://www.w3.org/voice

[3] W3C Working Draft, "Voice Extensible Markup Language (VoiceXML) Version 2.0", http://www.w3.org/TR/2002/WD-voicexml20-20020424/, 24 April 2002

[4] W3C Working Draft, "Voice Browser Call Control: CCXML Version 1.0", http://www.w3.org/TR/2002/WD-ccxml-20020221/, 21st February 2002

[5] W3C Working Draft, "Semantic Interpretation for Speech Recognition", http://www.w3.org/TR/2001/WD-semantic-interpretation-20011116/, 16 November 2001

[6] W3C Candidate Recommendation, "Speech Recognition Grammar Specification Version 1.0", http://www.w3.org/TR/2002/CR-speech-grammar-20020626, 26 June 2002

[7] W3C Working Draft, "Speech Synthesis Markup Language Specification", http://www.w3.org/TR/2002/WD-speech-synthesis-20020405/, *5 April 2002*

[8] W3C, "Multimodal Interaction Activity", http://www.w3.org/2002/mmi/

[9] Stéphane H. Maes (smeas@us.ibm.com) - IBM, and Chummun Ferial (chummun.ferial@ipce.eu.sony.jp) - Sony, "Multi-Modal Browser Architecture",

[10] SALT Forum Technical With Paper, "Speech Application Language Tags (SALT)", http://www.saltforum.org

[11] SALT Forum, " Speech Application Language Tags (SALT) – Specification 1.0", 15 July 2002

[12] Hitesh Seth, "SALT by Example", VoiceXMLPlanet.com, http://www.voicexmlplanet.com/articles/salt1.html

[13] Brian Eberman, Jerry Carter, Darren Meyer, and David Goddeau, "Building VoiceXML Browsers with OpenVXI", WWW2002, May 7-11, 2002, Honolulu, Hawaii, USA, http://www2002.org/CDROM/refereed/260/

[14] SpeechWorks International, Inc., "OpenVXI", http://www.speech.cs.cmu.edu/openvxi/

[15] Carnegie Mellon University, "OpenSALT", http://www.speech.cs.cmu.edu/OpenSALT

[16] Carnegie Mellon University, "Sphinx", http://www.speech.cs.cmu.edu/sphinx

[17] "Festival Speech Synthesis System", http://www.festvox.org

[18] Aljoša Jakovčić,"Raspoznavanje govora primjenom skrivenih Markovljevih modela", Faculty of Electrical Engineering, Zagreb

[19] Albert Novak, "Raspoznavanje govora uporabom skrivenih markovljevih modela i teorije valića", Faculty of Eletrical Engienering, Zagreb, 2002.

[20] Lawrence Rabiner, Biing Hwang Juang: ”Fundamentals of Speech Recognition”, Prentice-Hall, inc., 1993.

[21] Donald A. Cole, Joseph Mariani, HansUszkoreit, Aunic Zaenen, Victor Zue: “Survey of state of the Art in Human Language”. http://cslu.cse.ogi.edu/HLTsurvey/

[22] S. Young, D. Kreshow, J. Odell, D. Ollason, V. Voltchev, P. Woodland: “The HTK Book”, Cambridge University, 1999.

[23] Albert Novak, "Sustavi znanja u raspoznavanju govora", Faculty of Electrical Engineering, Zagreb, 1999.