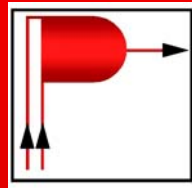


PROLOGIX

Software Solutions Pvt. Ltd.



VoiceXML

a White Paper

Prologix Software Solutions Pvt. Ltd.

8th Km., Faizabad Road,

Ismailganj, Lucknow – 226016,

India

www.prologixsoft.com



Table of Contents

1	Introduction	3
2	The VoiceXML Forum	3
2.1	What is VoiceXML?	4
2.2	Why VoiceXML?	4
3	Voice-Enabling the web	4
3.1	Candidates for Voice Application	5
4	Architectural Model of VXML	6
4.1	Goal of VoiceXML	7
5	Voice XML – the structure	7
6	The Prologix VoiceXML Interpreter	11
6.1	Architecture	13
6.2	The product that we are offering	14
6.3	Product highlights	14
6.4	VoiceXML tags supported by the current version:	15



1 Introduction

The Web's capabilities have grown rapidly since its inception.

Gone are the days when browsers used to display only static texts. The present day browsers support rich text, pictures, graphics, video, sound and can be said to have true Multimedia capability. Web browser user interface capabilities have expanded beyond simple link following to sophisticated user interfaces featuring forms, frames, client-side scripting, and Java applets.

Web servers' capabilities have grown too. Web pages, formerly static, are now commonly generated on demand from templates, programs and data. Database access is integrated into this process, as is access to legacy systems.

Bottlenecks in the Internet infrastructure are also getting reduced day by day. This has led to the emergence of many new types of web applications and services, which in turn accelerate further improvements in infrastructure. Today's web development tools are much efficient and are more powerful. The Internet has begun to extend beyond computers to other devices like personal organizers with wireless data connections and mobile phones. The future will bring more web devices to users at home and to businesses.

In this scenario, HTML, which was started out a long time ago as a contextual language for viewing scientific documents, tells only how the data should look and not what it means. It loses information about the web (its environment) and hence is not well suited for creating the interactive advertisements, gallery walk-throughs and online super-stores of the present day. However, we're on the edge of a revolution that will prove to be as profound as the Internet revolution in its own way. The forerunner of this new revolution is XML.

Extensible Markup Language (XML) is the universal language for data on the web. XML is a general and highly flexible representation of any type of data, be it plain text, audio or for that matter video, and the various transformation technologies make it easy to map one XML structure to another, or to map XML into other data formats. XML allows the creation of unique data formats for specific applications. It is also an ideal format for server-to-server transfer of structured data. XML specifies neither semantics nor a tag set. In fact XML is really a meta-language for describing markup languages.

Speech technology, as it improves, will become a very natural and powerful interface for the ubiquitous web devices. Microphones are much smaller than keyboards and keypads; speakers are smaller than screens. So it seems quite likely that many future web devices will have on-board speech recognition (as do some mobile phones today), or perhaps that we'll carry voice-activated universal remotes to talk to the devices in our immediate surroundings.

The need of the hour is to have a language, which can enable a user to interact orally with the web from any part of the globe. Enter VoiceXML - the Voice Extensible Markup Language. VoiceXML is based on XML and strongly benefits from the ability to move audio data efficiently across the web. It was designed for the development of speech based telephony applications on the web.

2 The VoiceXML Forum

AT&T, IBM, Lucent and Motorola founded the VoiceXML Forum, an industry consortium of over 300 companies. It was established to develop and promote VoiceXML, a computer language designed to make Internet content and information accessible via voice and phone. With the backing and technology contributions of its four world-class founders, and the support of leading Internet industry players, the VoiceXML Forum has made speech-enabled applications on the Internet a reality. For more information on the VoiceXML Forum please visit the website at <http://www.voicexml.org/>



VoiceXML 1.0 is a specification of the VoiceXML Forum, The Forum is active in the conformance testing, education, and marketing of VoiceXML, and has given control over further language development to the World Wide Web consortium (W3C). Because it is a specification, applications that work on one conformant VoiceXML platform will work on others as well. The official draft for VoiceXML 2 is already under review and could lead to a published specification soon.

2.1 What is VoiceXML?

VoiceXML (or simply VXML) stands for Voice Extensible Markup Language.

Much as HTML is a markup language for creating distributed textual/visual applications, VoiceXML is an XML based markup language for creating distributed voice applications. Using VoiceXML one can develop web-based applications that users can access from a telephone - using their voice. Simply by talking to a VoiceXML application, a user can call into a web site and access the very same information and services, instead of having to be logged into the Internet and be tied to a computer and a mouse.

2.2 Why VoiceXML?

By providing voice access to a web site over the phone, customers are provided with anytime anywhere access to the services without having to be tied to a computer and be logged into the web to access these services. With an increasing penetration of land line and mobile telephones in today's world, people essentially have a voice based access device with them at all times. Using VoiceXML one can thus provide access to web content to anyone at any remote place across the globe.

3 Voice-Enabling the web

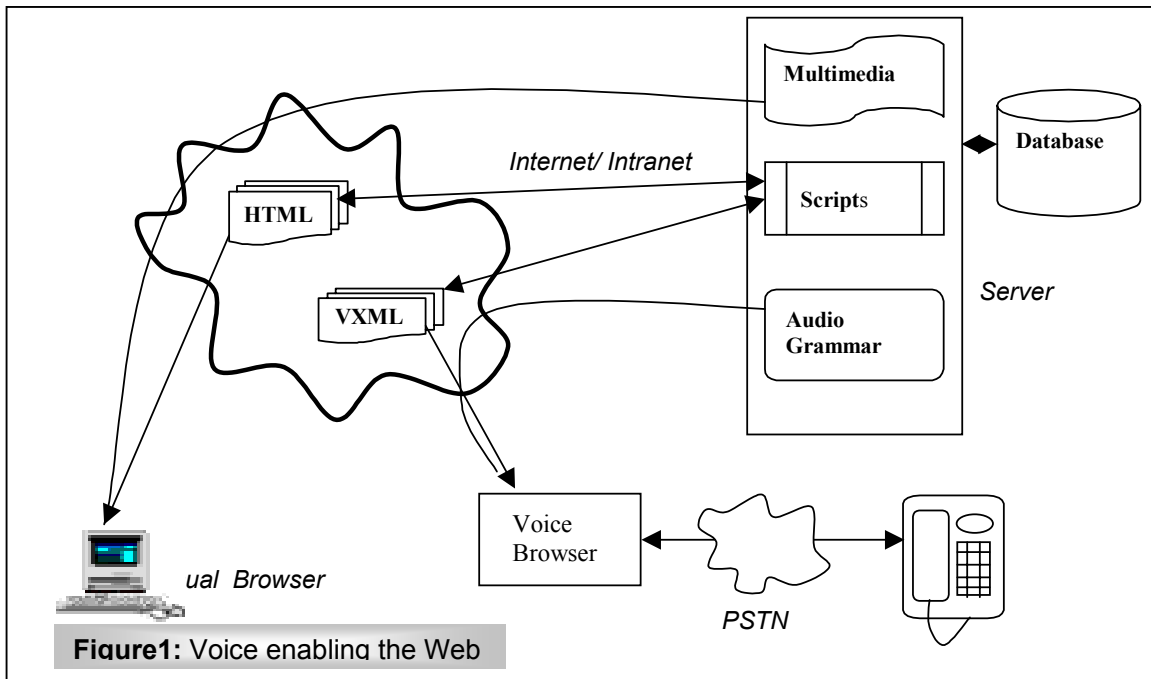
The term Voice-enabling the web does not imply reading textual information from the site and filtering out the graphics and other multimedia. Voice-enabling the web refers to a different way to access web content.

In today's scenario we use a GUI web browser (Visual Browser), which renders and interprets HTML to present information to the user (text, graphics, audio, hyperlinks, etc.). When the user makes a selection (for example, a click on a hyperlink), the GUI web browser sends an HTTP request to the web server (in this case, to retrieve another page). The web server responds by locating the new page and returns HTML to the browser to present the new page to the user. The web server may also have to interact with a back-end infrastructure (database, servlets, etc.) to obtain and return the requested information.

The Voice Browser is just an extension of this paradigm.

In Figure 1, a telephone, a Voice Browser and a Voice Server (Embedded in the Web server) has been added to the web environment. The Voice Server manages several Voice Browser sessions (one session per call), Each Voice Browser session includes an instance of the Voice Browser, the speech recognition engine, and the text-to-speech engine.

VoiceXML introduces a new way of presenting the same web information. Voice Browser presents the information to the caller by speaking out the prompts in the VoiceXML documents using the text-to-speech engine, instead of presenting the information visually (through HTML, graphics, and text). When the caller says something (which is the voice equivalent of clicking on something to make a selection), the Voice Browser uses voice recognition resources to interpret the voice input and sends an HTTP request to the web server (also called the document server in VoiceXML jargon), which may access the same back-end infrastructure, to return information as a VoiceXML document that could be rendered back to the user over the phone.



This Voice based interface need not always have an upper hand over the traditional visual one. There are some applications that are just more suited for a visual medium. For example, it may be quite acceptable to purchase a music CD or a book over the phone, but it may not be a good idea to use VXML to script an application that gives out some essentially visual information, say a picture gallery. So, what one has to do as an application designer is to decide on what kind of information, how much information, and how and when to present it to the user.

3.1 Candidates for Voice Application

Applications that can be voice enabled fall into two categories.

- Query applications

- Transaction applications

A customer calls a query application to retrieve information from a web database.

A transaction application is one into which the customer calls with the intention of performing specific transactions with a web-based back-end.

Some specific applications are being discussed herewith:

Applications for Information retrieval are good candidates that can be Voice -enabled. In these, the output tends to be pre-recorded information, and voice input is often constrained down to a few navigation commands and limited data entry (e.g., a few commands to browse a set of news articles, or the collection of zip code for a weather forecast). Information retrieval applications can provide news, sports, traffic, weather, and stock information, as well as more specialized information (e.g., intranet-based company news).

Electronic commerce is another area. Catalog ordering applications have to be done right, because voice conveys much less information than graphics, and that information must be serialized into a single audio stream. Customer service applications (package tracking, account status, and call centers) are well suited. Financial applications - banking, stock quotes and trading, e.g. - can work well too.



Telephone services like personal voice dialing and teleconference room setup and management can be voice-enabled through VoiceXML. An organization can upload up a voice web site to its voice service provider with information, news, upcoming events, and an address book. The address book could be used in voice dialing by the people in that organization. The voice service provider can charge a monthly hosting fee, or sell the address list.

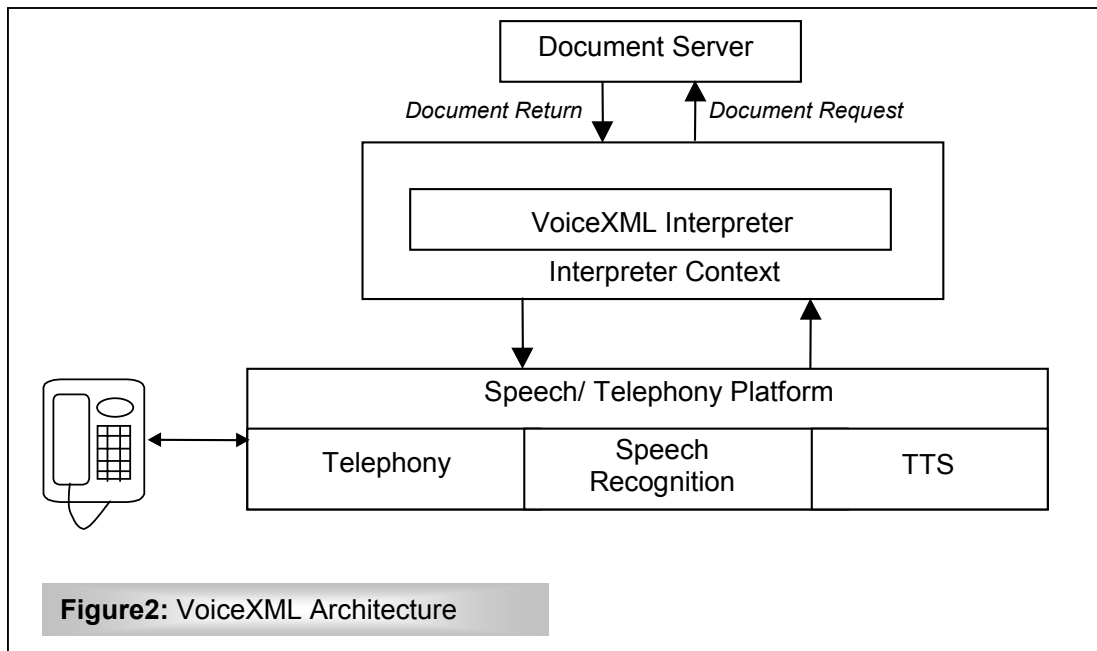
Because standard Web security features apply to the voice web, **intranet applications** can also be written in VoiceXML for inventory control, ordering supplies, providing human resource services, and for corporate portals.

Unified messaging applications can leverage voice. E-mail messages can be read over the phone, outgoing e-mail can be recorded or transcribed (when a large vocabulary voice recognition engine is available) over the phone, and voice-oriented address information can be synchronized with personal organizers and e-mail systems. Pager messages can be originated from the phone, or routed to the phone.

There are many other areas where voice services will be used, such as checking the status of bids at electronic auction sites, bill payment authorization, charitable goods pickup scheduling, wake up reminder services, and others we can't conceive of. And while all VoiceXML services will benefit visually impaired people, it may be that other services will be specially crafted for this community.

4 Architectural Model of VXML

The architectural model of the Voice browser assumed by VoiceXML Specification v1.0 is represented in Figure 2 below.



When a call is received, the Implementation platform sends an event to the VoiceXML interpreter, which in turn looks in its context for the URI of the initial document to fetch. A request is then sent to the document server for the initial document. The document server in turn sends the document to the VoiceXML Interpreter that interprets the document and executes it appropriately using the services of the speech/telephony platform. The interpretation may result in a message to be prompted to the caller through the implementation platform or the VoiceXML Interpreter making additional document requests



to the document server. Under the document's control, VoiceXML Interpreter directs the implementation platform to:

Send prompts, messages, or other audio material (say music or sound effects) to the user.

Accept numeric input that the user enters by DTMF (telephone key tone) signals.

Accept voice input and recognize the words by receiving speech recognition grammar data dynamically.

Simply record a voice input, without trying to recognize any words.

Send the user's information to a Web site or other Internet server.

Receive information from the Internet, and pass it to the user.

For instance, in an interactive voice response application, the VoiceXML interpreter context may be responsible for detecting an incoming call, acquiring the initial VoiceXML document, and answering the call, while the VoiceXML interpreter conducts the dialog after answer. The implementation platform generates events in response to user actions (e.g. spoken or character input received, disconnect) and system events (e.g. timer expiration).

Some of these events are acted upon by the VoiceXML interpreter itself, as specified by the VoiceXML document, while others are acted upon by the VoiceXML interpreter context.

4.1 Goal of VoiceXML

VoiceXML's main goal is to bring the full power of web development and content delivery to voice response applications, and to free the authors of such applications from low-level programming and resource management. It enables integration of voice services with data services using the familiar client-server paradigm. A voice service is viewed as a sequence of interaction dialogs between a user and an implementation platform. Document servers provide the dialogs, which may be external to the implementation platform. Document servers maintain overall service logic, perform database and legacy system operations, and produce dialogs. A VoiceXML document specifies each interaction dialog to be conducted by a VoiceXML interpreter. User input affects dialog interpretation and is collected into requests submitted to a document server. The document server may reply with another VoiceXML document to continue the user's session with other dialogs.

The specification claims VoiceXML as a markup language that:

- 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 (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.

While VoiceXML strives to accommodate the requirements of a majority of voice response services, services with stringent requirements may best be served by dedicated applications that employ a finer level of control.

5 Voice XML – the structure

VoiceXML code is contained in a document (just like HTML code is contained in a page). A VoiceXML application is a collection of VoiceXML documents just in lines with an HTML application, which is really just a collection of HTML pages.



Documents are transitioned to by specifying a Universal Resource Indicator, or URI (just like HTML pages are linked to by specifying a URL). The main document is the current (active) VoiceXML document.

The documents in a VoiceXML application share the same application root document. The application root document is loaded whenever any document within the application is active. This is the way to identify global scope (including variables and grammars).

VoiceXML is an XML schema. Based on the XML tag/attribute format, the VoiceXML syntax involves enclosing instructions (items) within a tag structure in the following manner:

```
< element_name attr="attr_value">
```

```
.....contained items.....
```

```
< /element_name>
```

A VoiceXML application consists of one or more text files called documents. These document files are denoted by a ".vxml" file extension and contain the various VoiceXML instructions for the application. It is recommended that the first instruction in any document to be seen by the interpreter be the XML version tag:

```
< ?xml version="1.0"?>
```

This tag encloses the vxml tag. The vxml element is the top-level element and encloses the entire document. The vxml tag with the version attribute set equal to the version of VoiceXML being used ("1.0" in the present case) is as follows:

```
< vxml version="1.0">
```

The document, which is contained in the vxml element, is split into dialogs. Form (`<form>`) and menu (`<menu>`) elements are the two types dialogs. Forms present information and gather inputs from the user. Menus offer choices of what to do next. A dialog presents information to the user using the `<prompt>` element or by simply a set of CDATA sections.

Here is a sample "Hello world" application:

```
<?xml version="1.0"?>
```

```
<vxml version="1.0">
```

```
  <form>
```

```
    <block>Hello World</block>
```

```
  </form>
```

```
</vxml>
```

When the above document is interpreted the user can hear "Hello World" spoken out by the TTS over the phone.

A form (or menu, considered to be a form with a single field) will contain various elements performing the tasks required by the form. Among the possible elements that can be contained in a form are those designated as "form items."

Form items are seven in number and come into either one of Field items or control items. To mention them,

Field items:

`<field>` - gathers input from the user via speech or DTMF recognition as defined by a grammar

`<record>` - records an audio clip from the user



`<transfer>` - transfers the user to another phone number

`<object>` - invokes a platform-specific object that may gather user input, returning the result as an ECMAScript object

`<subdialog>` - performs a call to another dialog or document(similar to a function call), returning the result as an ECMAScript object

Control items:

`<block>` - encloses a sequence of statements for prompting and computation

`<initial>` - controls mixed-initiative interactions within a form

Apart from these form items, a dialog is managed through the use of grammars (`<grammar>` or `<dtmf>`, which define what the user can say or enter using a telephone key pad, prompts `<prompt>` (which present information and instructions to the user), events (handling "non-normal" input), and control elements such as goto's `<goto>`, links `<link>`, submits `<submit>`, and the like.

Here is another dialog which prompts the user for the credit card number and submits the same to a card verification server script.

```

<form id="getCreditCardNumber">
  <field name="ccNo">
    <prompt>What's your card number?</prompt>
    <grammar src=" ../grammars/ccNo.gram" type="application/x-jsgf" />
    <help> Please say your six digit credit card number on the top left of your card something
      like one four one two six one</help>
  </field>
  <filled namelist="CCNumber">
    <prompt>Kindly hold on till your number is validated</prompt>
  <submit      next="http://www.ccno_verification.eg/ccno.asp"      namelist="ccNo"
  method="GET"/>
  </filled>
</form>

```

A sample interaction would go like this:

C: What's your card number?

H: One hundred forty five

C: I did not understand what you said. Ask for help if necessary

(Platform specific message...)

H: help

C: Please say your six digit card number on the top left of your card something like one four one two six one

H: one nine six eight seven four

C: Kindly hold on till your number is validated

C: (...Continues in ccno.asp...)



The dialog `getCreditCardNumber` (specified using the `id` attribute of the form) has a field item name `ccNo`. The field is managed by the grammar `<grammar>` specified in `ccNo.gram`. Grammars can be put into three classifications as:

- Built in Grammars
- Inline Grammars
- External Grammars

The grammar specified in the above dialog is an external grammar. Here the grammar can be specified anywhere in the web's sphere and can be linked to by specifying the URI of the grammar as a value to the `src` attribute of the corresponding `<grammar>`. Inline grammars are specified as CDATA sections, which are child of the enclosing `<grammar>` or `<dtmf>` tags. Built-in grammars are specified using the `type` attribute of the corresponding field element. The specification speaks of boolean, digits, date, currency, phone, time and number as the candidates for built-in grammars.

A field item to collect a phone number can be written as:

```
<field name="getPhoneNumber" type="phone">
<prompt>.....</prompt>
</field>
```

Next comes the filled (`<filled>`) element, which specifies the action to be executed when field elements are filled some user utterance or dtmf input. The `namelist` attribute of the `<filled>` specifies the names of the fields that should be filled in before it can be executed. In the example above for getting the CC number when a valid credit card number is uttered by the user the field `ccNo` is filled with the user input and this triggers the filled element. The filled element contains a `<submit>` that is used to submit values to document server using the conventional HTTP GET or POST method. The control then transfers to the document returned by the server in response to the submit.

VoiceXML allows for variables and Expressions as part of the document. Variables are declared using the `<var>` element. The `<assign>` element assigns value to a variable and its value when used subsequently in prompts is spoken out using the `<value>` element. A variable can be scoped according to its occurrence in the document. A variable can be declared in the following scopes:

Session - One declared by the interpreter Context and alive for the entire session

Application - Variables in the document scope of the application Root document

Document - Declared as children of a document's `<vxml>` element

Dialog - Declared as a child of the corresponding `<form>` or `<menu>`

(Anonymous) - Each `<block>`, `<filled>` and `<catch>` element can define this scope to contain the variables declared in the them.

To handle run time environments in an orderly fashion VoiceXML supports various events and their handling.

The platform throws events when the user does not respond, doesn't respond intelligibly, requests help, etc. The interpreter throws events if it finds a semantic error in a VoiceXML document, or when it encounters a `<throw>` element. Character strings identify events.

Each element in which an event can occur has a set of catch elements, which include:

```
<catch>, <error>, <help>, <noinput>, <nomatch>
```



An element inherits the catch elements ("as if by copy") from each of its ancestor elements, as needed. If a field, for example, does not contain a catch element for nomatch, but its form does, the form's nomatch catch element is used. In this way, common event handling behavior can be specified at any level, and it applies to all descendents.

`<if>`, `<else>`, `<elseif>` are the three elements utilized for conditional statements in VoiceXML. For eg:

```
<if cond="flavor == 'vanilla'">
<assign name="flavor_code" expr="v"/>
<elseif cond="flavor == 'chocolate'">
<assign name="flavor_code" expr="h"/>
<elseif cond="flavor == 'strawberry'">
<assign name="flavor_code" expr="b"/>
<else/>
<assign name="flavor_code" expr="?">
</if>
```

The `cond` attribute is used to specify the logical condition to be evaluated.

The `<goto>` element is used for unconditional transition to another form item in the current form, transition to another dialog in the current document, or transition to another document.

Apart from the dialogs, a document may also contain `<meta>` elements, `<var>` and `<script>` elements, `<property>` elements, `<catch>` elements, and `<link>` elements.

Fetching of content from a URI occurs in a VoiceXML interpreter context to: (1) fetch VoiceXML documents to interpret, or (2) fetch other document types, such as audio files, objects, grammars, and scripts. All occasions for fetching content in a VoiceXML interpreter context are governed by the three attributes: `catching`, `fetchhint`, `fetchtimeout` which are set using the `<property>` elements.

The VoiceXML interpreter context, just like HTML visual browsers, can use caching to improve performance in fetching documents and other resources; audio recordings (which can be quite large) are as common to VoiceXML documents as images are to HTML pages. The specification states the caching policy to be one that is commonly employed in HTML browsers.

This section has spoken in brief of the main composition of VoiceXML. The v1.0 specification of the VoiceXML forum dated 07 March 2000 can be referred to for a detailed study.

6 The Prologix VoiceXML Interpreter

The VoiceXML development group at Prologix has developed a highly scalable and robust implementation for VoiceXML. The interpreter has been developed in C++ language with an object-oriented design.

From a functional perspective Prologix's VoiceXML Interpreter is divided into the following major components that are independent of each other:

1. **VoiceXML interpreter** – Parses VoiceXML and executes appropriate functions when it requires performing some input/output. This is intended to be platform independent (the input/output takes place through a well defined interface which can be implemented over any specific platform).



2. **Platform** – This is a device/ appliance that provides the requisite voice resources such as text to speech, ASR and perhaps telephony resources. In case of a telephony platform, the call control and signaling may be conventional (TDM based) or IP based. The interpreter has a limited view of the platform and is concerned only with select functionality which it accesses through the input/output interface.
3. **HTTP module** – This is required for GET and POST of data for the Interpreter and is classified with the Platform.

The VoiceXML interpreter requires a Document Object Model (DOM) parse tree which can then be traversed appropriately (Interpreted and appropriate actions taken) and rendered to the platform by the interpreter context. An XML parser is required to generate the DOM parse tree. JavaScript is also an important component of the VoiceXML specification. All expressions to be evaluated by the Interpreter context are passed on to the JavaScript Engine which evaluates them in its own context and renders the result back to the interpreter context.

The Interpreter module in its own is platform independent. As stated earlier the implementation platform can be vendor specific. An application in that platform is to be written which can start the Interpreter with following function calls embedded into it.

The interpreter SDK can be used to easily embed the interpreter in any speech/telephony platform. To do this, the developer needs to:

- a) Implement the interpreter's input/output interface over the given platform. This simple and well-defined interface covers basic speech and telephony capabilities.
- b) Instantiate the InterpreterContext object and invoke the relevant methods as shown below.

```
/*Instantiate the VXMLInterface that holds the Handle to
JSEngine,TreeClass,HTTPInterface and the logger*/
VXMLInterface * vxmlIFace = new VXMLInterface(1024*1024*16,1,j,mtx);
/*Instantiate the InterpreterContext*/
InterpreterContext *ic = new InterpreterContext(vxmlIFace);
/*Set the VXMLFile to be parsed*/
ic->SetURI(ScriptFileName);
/*Start Interpretation*/
ic->Start();
/*Do clean up*/
delete ic;
ic = NULL;
delete vxmlIFace;
vxmlIFace = NULL;
```

The Interpreter expects the platform to provide implementation for speech recognition/synthesis, playing audio files and for collecting DTMF input. The input/output interface implementation takes care of this. Prologix provides comprehensive documentation with clearly commented sample source code to help the customer understand and



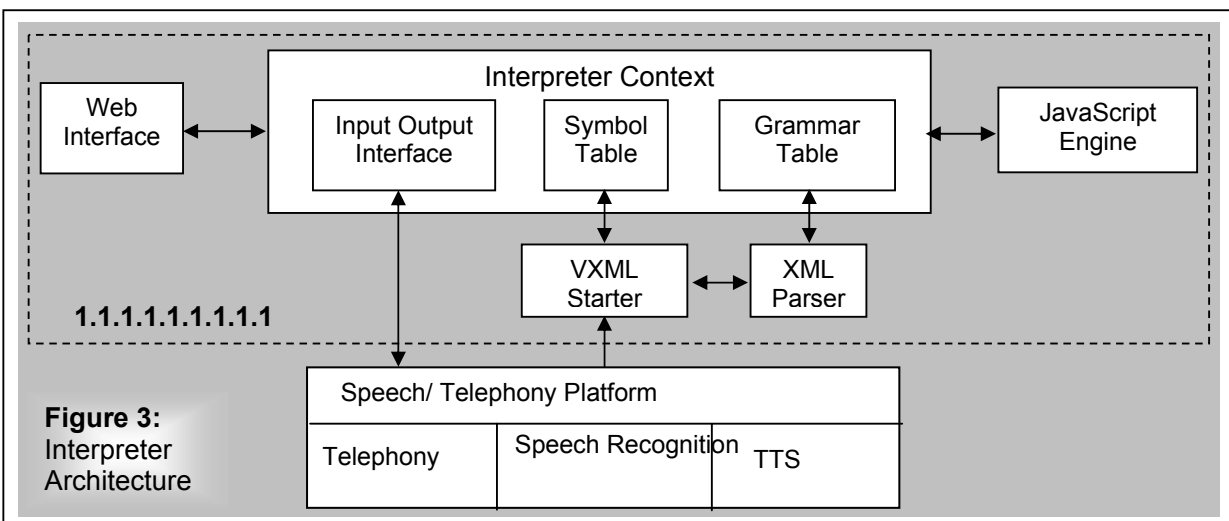
implement the input/output interface. Alternately, Prologix can also provide professional services to carry out this implementation.

The HTTP module acts as an interface between the interpreter and the web. Its main purpose is to implement the HTTP GET and POST methods. The implementation of this class is OS specific.

6.1 Architecture

The interpreter has been designed to meet the following architecture goals:

1. **Platform independence:** *The interpreter should have minimal dependence on a given speech/telephony platform and should be easily portable across platforms.* The insulation between core interpreter features and platform specific features has been guaranteed by encapsulating the platform specific features in a well-defined interface and allowing only the implementation of this interface to be platform specific.
2. **Modularity:** The VoiceXML standard is still some kind of a moving target. A Version 2 draft is already available and brings to light some proposed changes in tag names, new tags and features. *The interpreter should implement related features in independent blocks or layers so that modularity can be leveraged to produce a compliant version soon after a new version of VXML becomes available.*
3. **Scalability:** *The interpreter should be scalable.* The product targets large voice portal companies and customer installations running several thousand VXML ports. The core interpreter (VoiceXML parser and VoiceXML interpreter context) has been implemented as a thread safe module and multi-threaded execution occurs during operation (usually every channel runs on a separate thread). This results in the interpreter overhead being extremely small. Further scalability may be achieved by implementing interpreter and platform in a distributed environment (on separate boxes) with the interpreter and application server running on a separate box. Since the interpreter manages all platform specific interactions through the input/output module, by implementing the i/o module over a distributed application protocol (such as CORBA or an IP telephony protocol like SIP) this can be achieved with ease.
4. **High performance:** Large VoiceXML applications, involving heavy multimedia exchange between client server applications, could consume computing resources and result in slow, sluggish applications. *The interpreter has been designed for high performance by use of efficient asynchronous communication between modules.* Further, VoiceXML allows platform specific functionality to be available to the interpreter users through `<object>` tag. Prologix's interpreter allows a unique plug-in design that allows developers to build high performance, platform specific objects (for instance ODBC) and plug these into the interpreter after optimization and testing. This is a powerful approach and allows the interpreter to take advantage of a high performance platform/engine that lies below it.





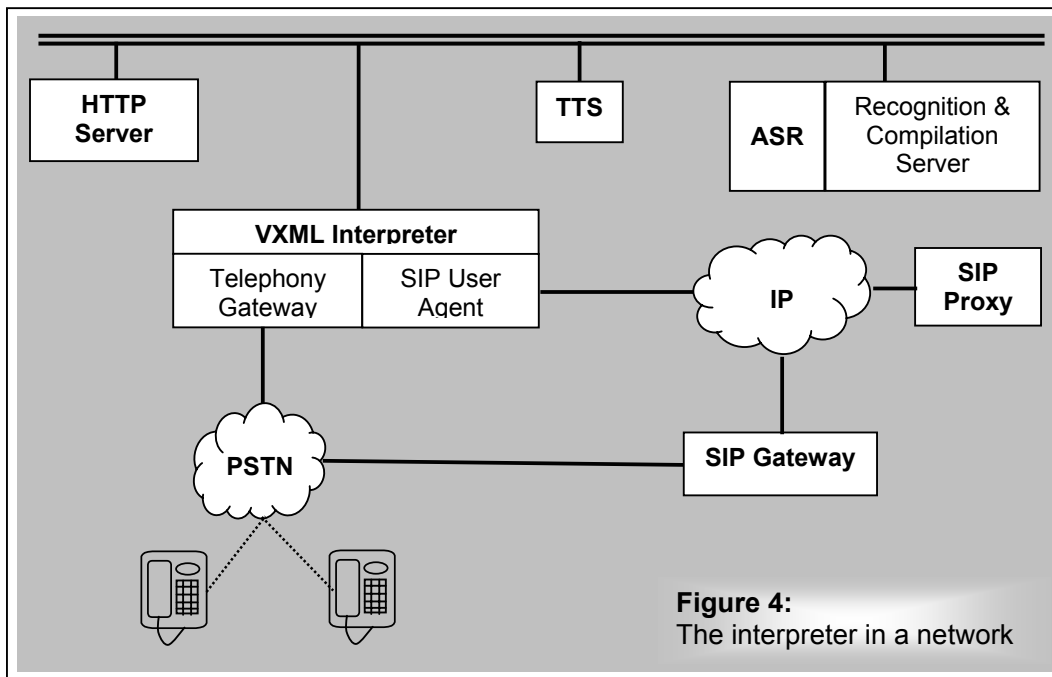
6.2 The product that we are offering

Our product is a VoiceXML browser that fully complies with the VXML standard (see salient features). We can provide this software in the following possible ways:

Complete solution: We can construct a complete solution for the customer. This will consist of a telephony component (call control and voice processing platform such as Dialogic), an advanced speech recognition engine (such as Nuance ASR), a Text to speech engine (such as Elan's TTS) and our VoiceXML interpreter. We can create and configure an optimized system using these components. All except the VXML interpreter will be bought outs.

Interpreter only: It is possible that the customer has made investments in TTS and ASR components already. For such customers, we can provide our VoiceXML interpreter in the form a software development kit (SDK) that they can compile and link with their existing software. We can even assist them in this integration process. Finally, we are also open to discussing source code sale for the interpreter.

For each of the two models above, we can provide a per voice port pricing. We can structure the voice port pricing for different slabs (low, medium and volume).



6.3 Product highlights

Some salient features of our VXML interpreter are as follows:

- **Full compliance with VXML 2.0 (October 2001).** We are closely tracking changes in the draft for version 2 and will be compliant in a very short time after the specification is released.
- **Highly scalable, portable C++ implementation.** The interpreter design abstracts all call control and I/O related calls, events and handlers in one class. It is easy to change or port to another platform, TTS, ASR and even OS.
- **High performance.** The interpreter overhead is very low. For medium to high density boxes running application and call control on the same box, we can guarantee 120 channels on a dual processor Intel system. We are further optimizing the design for Compact PCI hardware and for operation in a distributed mode.



- **We support all built in grammars as defined by VoiceXML.** We also support Dynamic grammars.
- **Full support for mixed initiative dialogs.**
- **A unique feature is the ability to support custom objects.** We can write and bundle custom objects with the interpreter; further we have a unique mechanism that allows customers to build and plug in their own platform specific objects for use in the interpreter. This is a very powerful approach and expands the VXML platform's capability and usability enormously.
- **Sophisticated run time error handling and logging and support for debug mode operation.**

6.4 VoiceXML tags supported by the current version:

S.no	Tagname	Attributes not supported (if any)	Comments
1	assign	-	Supported
2	Audio	-	Supported
3	Block	-	Supported
4	catch	-	Supported
5	choice	-	Supported
6	Clear	-	Supported
7	Disconnect	-	Supported
8	Else	-	Supported
9	Elseif	-	Supported
10	Enumerate	-	Supported
11	Error	-	Supported
12	Exit	-	Supported
13	Field	-	Supported
14	Filled	-	Supported
15	Form	-	Supported
16	Goto	-	Supported
17	Grammar	Type,root,xml:lang	Partially Supported (Requires platform support)
18	Help	-	Supported
19	If	-	Supported
20	Initial	-	Supported
21	Link	-	Supported
22	Log	-	Supported
23	Menu	-	Supported
24	Meta	-	Not Implemented
25	Noinput	-	Supported
26	Nomatch	-	Supported
27	Object	-	Supported Note:- Objects have platform specific implementation. Prologix provides select objects for certain generic functionality only.
28	Option	-	Supported
29	Param	-	Supported



S.no	Tagname	Attributes not supported (if any)	Comments
30	Prompt	-	Supported Note:- Full support for Speech Markup(SAPI Compliant) is provided at the level of Interpreter itself. Thus Interpreter does not depend on TTS for speech markup support.
31	Property	-	Supported
32	Record	-	Supported
33	Reprompt	-	Supported
34	Return	-	Supported
35	Script	-	Supported
36	Subdialog	-	Supported
37	Submit	-	Supported
38	Throw	-	Supported
39	Transfer	-	Supported
40	Value	-	Supported
41	Var	-	Supported
42	Vxml	-	Supported

Summary of Implementation Status:

Tags Fully Implemented = 40

Tags Partially Implemented=1

Tags Not Implemented=1