



CONTENTS

Introduction 1

 Would you believe 1

 VoiceXML - A mixed blessing 2

Is an IVR really a Voice Browser? 3

 The origin of VoiceXML 3

 The “Voice Browser” Model 4

 The “IVR” Model 5

 Conflicting Goals 6

Complexity and the
VoiceXML Eco-system 6

The “Dark Side” of VoiceXML:
Performance 9

 Measuring Performance 9

 Our Performance Numbers 10

Porting Your Applications to SPOT 11

 How easy is it to Port? 11

Interact as your Technology Partner 12

Take our SPOT SIP Engine Out for a Run! 13

About the Author 14

Introduction

So, you’ve been sold on the virtues of standards, converted your voice applications to VoiceXML (VXML), and now your performance has tanked?

The SPOT SIP Engine gives you the best of both worlds: the performance of proprietary application generators, the rapid deployment, portability and ease of change with VoiceXML.

The SPOT SIP Engine is Interact’s “state-of-the-standards” voice application engine or platform featuring:

- a conformant VoiceXML 2.0 and 2.1 voice-browser/interpreter
- a conformant CCXML 1.0 Interpreter
- SIP based VoIP telephony (conformant with the SIP Forum’s SIPconnect 1.0 recommendation)
- Host Media Processing (all audio processing uses the host server’s CPU)
- Your choice of ASR/TTS engines via MRCP V2, if voice synthesis or recognition is desired
- built-in application redundancy support, so active calls can survive a server failure
- all on a Linux server, yet manageable remotely via a standard web browser on any OS

Would you believe:

1400 voice channels (ports), 200 calls-per-second, with 40% idle on a single server with dual quad-core CPUs, 2.3 GHz, for two different high stress applications on 32 bit Linux:

- **Whisper:** an application that simply answers a call, plays a 4 second prompt, and hangs up. Mobile carriers often use this type of application to thank a customer for using their service while the call is being placed.
- **VULT (VoIP Ultimate Load Test):** an application characterized as an inbound/outbound dialer with playback, sending/detecting DTMF, grammar matching, and a handful of web requests - as the name says, a total performance stress test.

To obtain this type of performance for your VoiceXML implementations, read on.

VoiceXML - A mixed blessing

Make no mistake about it, not only is VoiceXML here to stay, it is on its way to being the scripting language used in all IVRs and voice applications in the foreseeable future. According to DataMonitor, the cross over point where more VoiceXML IVR ports than proprietary IVR ports shipped occurred in 2008.

That said, VoiceXML is unfortunately both a service and a disservice to the Interactive Voice Response (IVR) industry per se.

On the positive front, the industry is slowly reaping the benefits of having an open standard:

- A common scripting language for voice applications supported by a variety of standards organizations (e.g., the W3C and IETF) and industrial consortiums (e.g., the 3GPP)
- Application portability (although that is proving to be somewhat ephemeral)
- The flip side of portability, vendor independence
- A cadre of trained professionals to support voice application development

On the other hand, the industry has been burdened with:

- An underlying “voice browser” model more suited to voice portals than to IVRs
- A complex, difficult to implement scripting language and eco-system
- A standard where compliance is antithetical to performance

It is this two-edged sword that led Interact to implement its conformant SPOT (Speech Processing of Tomorrow) SIP Engine.

Our interpreter implementation supports the creation and deployment of fully-featured voice applications, with lower CPU and memory requirements of other VoiceXML and CCXML interpreters.

In this white paper, we will:

- Contrast the underlying model espoused by the VoiceXML authors with that of an IVR
- Review VoiceXML and it’s associated eco-system
- Discuss performance from an IVR perspective and showcase SPOT’s numbers
- Discuss what’s involved in porting your VoiceXML application to a SPOT SIP Engine
- Conclude by summarizing SPOT’s design objectives and licensing opportunities

Is an IVR really a Voice Browser?

The origin of VoiceXML

Although answering machines, voicemail, auto-attendants, IVRs and Call Centers all pre-date the early '80s, there was an explosion in these and other voice technologies driven by the advent of Personal Computers and add-in telephony cards that interfaced with telephone lines.

In the '90s, several companies were developing proprietary "mark up languages", languages based on XML, for voice applications. AT&T, IBM, Lucent, and Motorola founded a cross industry VoiceXML Forum in early 1999, and a group of experts under the Forum's auspices put together a draft specification of a language based on the best features of their individual efforts, along with new concepts they jointly developed. In March 2000, they published the VoiceXML 1.0 specification, and then submitted it to the World-Wide Web Consortium (W3C) for consideration.

The W3C's Voice Browser Working Group (VBWG), which had come into existence in early 1999, took on the task of the next revision, and in 2001 the W3C published initial drafts of three additional mark-up languages along with a draft for VoiceXML 2.0 for specifying conversational dialogs in which callers speak and listen to speech-enabled applications.

The complementary markup languages were:

- Speech Recognition Grammar Specification (SRGS),
- Speech Synthesis Markup Language (SSML).
- Semantic Interpretation for Speech Recognition (SISR)

In February 2004 the W3C published a draft of a set of eight additional features added to VoiceXML 2.0, called VoiceXML 2.1. No changes are necessary to VoiceXML 2.0 applications to run under VoiceXML 2.1.

In an attempt to maintain the purity of VoiceXML as a "presentation layer" language, the VBWG also created a draft specification in 2001 for Call Control XML or CCXML, for handling the telephony session control.

Although initial drafts were published in 2001, the W3C specification process is glacially slow and has a multitude of steps aimed at building a consensus through a sequence of drafts before one gets to an actual recommendation. (See <http://www.w3.org/2005/10/Process-20051014/tr> for details of the process, if you are interested.)

Of this eco-system of inter-related standards, collectively referred to as the W3C Speech Interface Framework, VoiceXML 2.0 along with SRGS and SSML finally became recommendations in 2004, SISR and VoiceXML 2.1 became recommendations in April and June respectively of 2007 and CCXML is anticipated to become a recommendation in 2011.

The “Voice Browser” Model

The abstraction being used to model voice applications was that of the “voice browser”, arising from the traditional W3C web browser model. Your computer has a browser that connects to a web server. That server presents content to you by providing HTTP to your browser, which formats the information for you. Behind the web server’s presentation layer is an application layer where all the logic of an application resides, along with the databases.

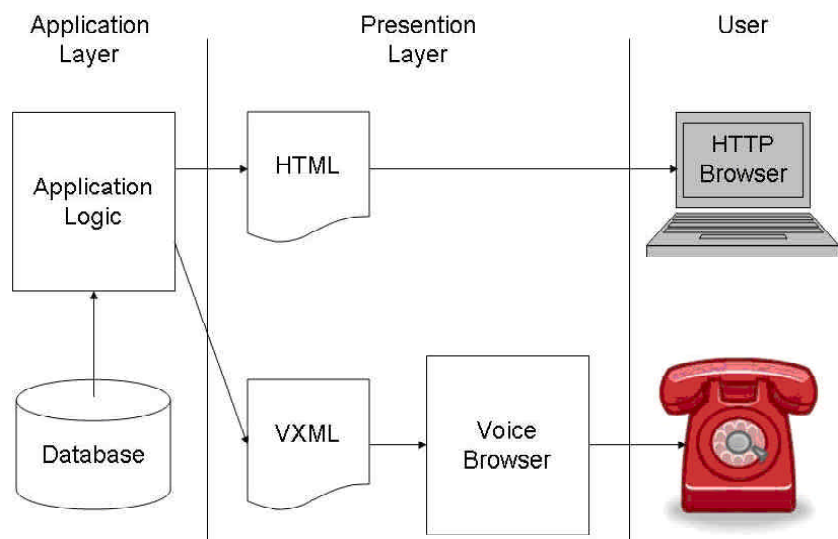
In the “voice browser” analogy, VoiceXML is the analog of HTTP, and specifies the dialog with the user. But since the traditional POTS telephone has no computational capability, the “browser” itself had to reside in the presentation layer. See the diagram below.

From the Introduction on the VBWG’s Voice Browser Activity page (<http://www.w3.org/Voice/>):

“The convergence of telecommunications and the Web is now bringing the benefits of Web technology to the telephone, enabling Web developers to create applications that can be accessed via any telephone, and allowing people to interact with these applications via speech and telephone keypads. The W3C Speech Interface Framework is a suite of markup specifications aimed at realizing this goal. It covers voice dialogs, speech synthesis, speech recognition, telephony call control for voice browsers and other requirements for interactive voice response applications,”

And from the Q & A section, in response to why the VBWG is so interested in VoiceXML 2.0:

“W3C’s mission is to fulfill the potential of the Web. With an estimated 2 billion fixed line and mobile phones world-wide, VoiceXML will allow an unprecedented number of people to use any telephone to interact with appropriately designed Web-based services via key pads, spoken commands, listening to pre-recorded speech, synthetic speech and music.”



Browser Model

The “IVR” Model

Most people’s exposure to Voice Applications comes from direct interactions with and use of: Automated Attendants, Voicemail, and more recently with IVRs used to front-end Call Centers for the Health, Insurance, Credit Card, Utility, Banking, Travel and many other industries.

Almost every definition of an IVR defines it in terms of a phone technology that allows a computer to detect voice and touch tones using a normal phone call.

- Wikipedia (www.wikipedia.com) An IVR is
“a phone technology that allows a computer to detect voice and touch tones using a normal phone call. The IVR system can respond with pre-recorded or dynamically generated audio to further direct callers on how to proceed. IVR systems can be used to control almost any function where the interface can be broken down into a series of simple menu choices. Once constructed IVR systems generally scale well to handle large call volumes.”
- Webopedia (www.webopedia.com) An IVR is
“a telephony technology in which someone uses a touch-tone telephone to interact with a database to acquire information from or enter data into the database. IVR technology does not require human interaction over the telephone as the user’s interaction with the database is predetermined by what the IVR system will allow the user access to. For example, banks and credit card companies use IVR systems so that their customers can receive up-to-date account information instantly and easily without having to speak directly to a person. IVR technology is also used to gather information, as in the case of telephone surveys in which the user is prompted to answer questions by pushing the numbers on a touch-tone telephone.”
- ZDNet (www.zdnet.com) An IVR is
“an automated telephone information system that speaks to the caller with a combination of fixed voice menus and data extracted from databases in real time. The caller responds by pressing digits on the telephone or speaking words or short phrases. Applications include bank-by-phone, flight-scheduling information and automated order entry and tracking. IVR systems allow callers to get needed information 24 hours a day. They are also used as a front end to call centers in order to route as many calls as possible away from costly human agents. In such cases, IVR does not replace the agent, but keeps them from constantly having to answer the same simple questions. Most IVR systems reside in PCs equipped with specialized PCI cards that connect to the telephone system to switch the calls.”

Common throughout all are these notions:

- A computerized telephony system
- Offloads the need to have a live agent at the other end of the conversation
- Structured “self-service” applications
- Structured front ends to call centers handling initial data collection

And IVR applications themselves may be classified as:

- In-bound: typically self service applications where the user initiates the call and the IVR provides the requested information
- Out-bound: the IVR initiates calls to users, (e.g., notifications, infomercials and requests for action)
- Bridging: where an inbound call results in an outbound call being placed, and if answered the called individual is bridged to the caller, for example PBX auto-attendant, or transferring to an agent in a call center

Conflicting Goals

When we look at these two models, the differences show up in sharp relief:

- The VBWG focus is the ability to interact with web pages from a variety of web sites.
- The IVR focus is mitigating the costs associated with having live agents running interference between a user and the enterprise's database.

Yet except for a handful of "café applications" – sites where a user can upload and/or generate VoiceXML scripts and test them by phone (PSTN or VoIP), the dominant use of VoiceXML today is in IVR applications. And aside from hobbyists and researchers, most café applications are aimed at hosted IVR solutions. Further, almost all the examples in the VoiceXML specifications are IVR applications.

Net, many of the design decisions in the creation of the language were based on a browser model that is inimical to efficiency (and therefore ultimately to cost) for the paramount application - usage in IVRs.

Some of this is even apparent in the 2006 VoiceXML Member Survey reported in the April May 2007 Issue of the VoiceXML Review (an e-zine published by the VoiceXML Forum), http://www.voicexml.org/Review/May2007/features/VoiceXML_survey.html, where one of the cited "perceived" weaknesses was the lack of "features to make it a full IVR/contact center programming language". The response to this and other perceived weaknesses deals with what is the scope of a markup language versus the application that generates it.

Complexity and the VoiceXML Eco-system

The W3C Voice Browser Working Group's Speech Interface Framework consists of a growing number of inter-related specifications (all at various stages in the process, from first draft, late draft, last call, and recommendation). This section briefly describes the various roles each has in creating Voice Applications. A list of the various specifications, and their current status in the W3C standards process, is available at (<http://www.w3.org/Voice/>)

All the specifications in the Speech Framework are XML-based application languages. XML (eXtensible Markup Language) is a general-purpose markup language - extensible as its users define their own tags - created to facilitate sharing of structured data across different information systems, particularly via the Internet. XML is used both to encode documents and serialize data in a human-legible manner. By adding semantic constraints, application languages can be implemented in XML.

The two key specifications, CCXML and VoiceXML 2.0 are designed to operate together, but each is independent of the other. That is, VoiceXML by design can be used with some other mechanism for initiating dialogs, and CCXML can be used with other dialog specification mechanisms. But this "design independence" has imposed a cost, as VoiceXML has some call control in the specification, and CCXML has some audio management in its specification. These ventures into the other's territory collide when both are actually used together.

The semantics of VoiceXML force an interpretive language. CCXML's designers were heavily focused on efficiency and deliberately avoided requirements that could only be implemented by interpretation or run-time evaluation.

- Voice Extensible Markup Language (VoiceXML) Version 2.0
W3C Recommendation 16 March 2004
<http://www.w3.org/TR/voicexml20/>
- Voice Browser Call Control: CCXML Version 1.0,
W3C Candidate Recommendation 1 April 2010
<http://www.w3.org/TR/2010/CR-ccxml-20100401/>

VoiceXML 2.1 provides a small set of additional features, with complete backwards compatibility with the VoiceXML 2.0 specification. This became a recommendation in June of 07.

- Voice Extensible Markup Language (VoiceXML) 2.1
W3C Recommendation 19 June 2007
<http://www.w3.org/TR/voicexml21/>

The W3C has a commendable philosophy of not reinventing the wheel, but referring to other Recommendations/Specifications/Drafts (theirs or others) where standardization either exists or is in process. To that end three additional VBWG specifications, each with different constituencies participating in the standardization process, are incorporated into the VoiceXML Recommendation, each in a different manner.

First, the VoiceXML 2.0 specification incorporates most of the SRGS specification, along with some augmentation, for use in inline grammars. It also allows external grammar files that follow the full SRGS specification. The grammar specification is used to specify the grammars that define acceptable responses whenever user input is requested (i.e. users are restricted to a specified response vocabulary when prompted for input - "Press or say one").

- Speech Recognition Grammar Specification (SRGS) Version 1.0
W3C Recommendation 16 March 2004
<http://www.w3.org/TR/speech-grammar/>

This works fine when the computer/IVR is directing the conversation. In an attempt to allow more general type conversations, VoiceXML provides for "Mixed Initiative" style conversations and Semantic Interpretations of user responses.

Semantic Interpretation means that the IVR system not only hears what the user says, but in a limited way defines what the user actually means. For example, if the system is looking for confirmation ("Yes"), it might allow the grammar to accept the usual alternatives that we humans accept for an affirmative response: "yeah", "yup", "sure", and "okay". The SISR specification embeds a semantic interpretation (the meaning of a response) in the grammar via tags. If there are tags, and the ASR Speech Engine employed supports those tags, then the VoiceXML interpreter has access not only to the user's actual utterance (in text form as recognized by the grammar), but also to its semantic interpretation.

- Semantic Interpretation for Speech Recognition (SISR) Version 1.0
W3C Recommendation 5 April 2007
<http://www.w3.org/TR/semantic-interpretation/>

Mixed Initiative allows the conversation between IVR and user to some extent to be guided by the user, rather than being simply a hierarchy of menu choices. The Holy Grail would be to prompt a user along the lines of "What would you like to do today?" but practicality requires a more relevant opening prompt. Mixed Initiative depends on Semantic Interpretation. For example, an initial prompt: "What would you like to order?" with a response: "I'd like three large pepperoni pizzas and a 2-liter bottle of coke." provides several data items at one fell swoop: quantity, size and type of pizza, and quantity, size and flavor of beverage. Mixed Initiative requires grammars with an embedded Semantic Interpretation.

Note however, neither Mixed Initiative nor SISR tags (i.e., semantic interpretation) are required in most IVR applications.

VoiceXML, IVR Performance and the SPOT SIP Engine

The remaining VBWG standard in the VoiceXML eco-system is designed to work with a Text To Speech (TTS) engine for dynamically generated spoken output

- Speech Synthesis Markup Language (SSML) Version 1.0
W3C Recommendation 7 September 2004
<http://www.w3.org/TR/speech-synthesis/>

This markup language is embedded in the actual text that is to be converted to speech, to guide the TTS Speech Engine. It permits selection of voice characteristics (name, gender and age) and control of speed, volume, pitch, and emphasis, as well as “say-as” capabilities like date, time, number ... Here only part of the specification is incorporated, and the use of standalone speech files has been omitted.

The use of SSML is not a required for TTS, but serves to make the TTS Engine “more human-like”.

Both the VoiceXML Forum and the W3C’s VBWG consider Media Resource Control Protocol (MRCP) a key enabling technology. MRCP specifies a common interface to Speech Engines for Automatic Speech Recognition (ASR) and TTS, and allows client devices, such as a voice browser, to interact with these resources in a standards-based, vendor-agnostic manner. The responsibility for MRCP V2 lies with the Speech Services Control (SpeechSC) Working Group of the Internet Engineering Task Force (IETF). That specification is built on SIP and RTP:

- Revision of Media Resource Control Protocol Version 2 (MRCPv2)
draft-ietf-speechsc-mrcpv2-24
<http://www.ietf.org/internet-drafts/draft-ietf-speechsc-mrcpv2-24.txt>

VoiceXML 2.0 is a declarative mark-up language that describes the user interface – the presentation layer for the exchange of information between a human and a computer for an interactive voice application. It is essentially a sequence of dialogs. CCXML is a markup language for controlling how (phone) calls are placed, answered, transferred, conferenced, etc. (The working group editor/chair calls it “session control for telephone calls”). CCXML operates as a state machine for a session based off states and events for transitioning between states. Both therefore provide only a basic computational capability with the <var> and <assign> elements, and then fall back to ECMAScript (JavaScript) for any advanced computation:

- Standard ECMA-262 ECMAScript Language Specification 3rd edition (December 1999)
<http://www.ecma-international.org/publications/standards/Ecma-262.htm>

although CCXML only mandates the so-called ECMAScript Compact Profile (ES-CP).

This rich eco-system, the complexity of the inter-related specifications and the interpretative nature of the core VoiceXML specification have all contributed to slow down the adoption rate of VoiceXML in the IVR industry:

- Implementations are costly, most early VoiceXML platforms, particularly production IVR platforms, have come from very large companies
- Voice engines (for TTS/ASR) have only recently been available via MRCP instead of having to be explicitly integrated
- Performance of most implementations is a major problem, driving up the need for hardware and cost per port

In spite of the attendant advantages of having open standards available for a good part of the last decade, as mentioned earlier the IVR industry only passed the break-even point in shipping proprietary vs. VoiceXML IVR ports in 2008.

The “Dark Side” of VoiceXML: Performance

IVRs initially were implemented in standard programming languages, and later some manufacturers shifted to proprietary scripting languages for application flexibility and performance. It was in the late '90s that VoiceXML itself grew out of the VoiceXML Forum's founders' desire for a standardized scripting or mark-up language.

And that's the downside or “dirty secret” of VoiceXML, a fully conforming implementation (see Appendix F of the VoiceXML V2.0 Recommendation, titled “Conformance”) cannot come near the performance of proprietary IVRs. Moreover, implementations do not easily scale. And that's tacitly admitted in the VoiceXML recommendation, as Appendix F contains the statement:

“There is, however, no conformance requirement with respect to performance characteristics of the VoiceXML Processor.”

On the other hand, the VBWG subcommittee working on the CCXML specification was quite concerned about performance, and in section 3.3 of the specification, call for the use of the “Compact Profile” version of ECMAScript (JavaScript), stating

“The constraints of ES-CP emphasize CCXML's ongoing concern for execution efficiency.”

ECMAScript Compact Profile (ES-CP) was initially designed for highly resource constrained battery-powered devices (think PDAs). Mandating only that subset emphasizes CCXML's concern for execution efficiency.

This “secret” is evidenced in several other ways:

- It is almost impossible to find published performance numbers for any VoiceXML vendor
- There are almost no published benchmarks for comparing VoiceXML implementations
- There are a number of articles on the Internet about how to exploit/fine-tune caching to increase VoiceXML performance

Measuring Performance

Performance of an IVR depends on several factors: channel capacity, application characteristics, call rate, and others.

Capacity for an IVR is usually measured in channels or ports - how many simultaneous user/IVR telephone conversations a given system can support? It's a function of available memory and processor cycles in the host server. In the TDM world capacity is capped by the physical constraint imposed by the number of bearer channels connected to the IVR. Ideally if there are 12 E1s connected to an IVR, capacity should be 360 - 372 channels (depending on signaling method).

With Voice over Internet Protocol (VoIP), where voice traffic is a stream going from one IP address to another, one can talk about half-duplex channels (one way audio), but for a dialog, you need to have one in each direction. We always assume full duplex channels in performance discussions.

Performance is measured in terms of how many calls-per-second (cps) an IVR can handle in some standardized application. A voice application takes anywhere from a few seconds (“Thank you for calling Interact Inc.”) to several minutes for an agent handled call, so for a given channel count, cps and the application characteristics must both be specified to measure comparative performance.

Dispensing with queuing theory, and simply multiplying calls-per-second by the average duration of a call for your application, you hopefully get close to the IVR's capacity, with lots of processor cycles (measured by % idle) remaining.

Our Performance Numbers

For the SPOT SIP Engine, we recommend the following minimums for general VoiceXML applications:

- For a 32 bit Linux system/server:
 - 3 MB RAM per channel, with a 512 MB RAM minimum
 - 13 MHz per channel, for up to 1400 channels.
 - 200 CPS maximum
- For 64 bit Linux system/server:
 - 3 MB RAM per channel, with a 512 MB RAM minimum
 - 19 MHz per channel, for up to 1000 channels
 - 200 CPS maximum

(Yes, using the 64 bit Linux OS is slower than the 32 bit.)

These minimums can be used as a rough first cut to determine what type of performance you would have using the SPOT SIP Engine for your VoiceXML applications.

For benchmarking performance we typically use a very simple voice application - but one that really stresses an implementation - a single VoiceXML document, answering a call and playing a 4 second pre-recorded prompt, both document and prompt from the host's file system. Although this benchmark may seem somewhat artificial, it is a typical IVR application in a carrier infrastructure, thanking a user for using that carrier's service during the time the subscriber is being connected to their called party. This benchmark really stresses the ability of the IVR to set up/tear down a call.

For loadtesting an IVR, we use an internal application called VULT (VoIP Ultimate Load Test) which is an inbound/outbound dialer with playback, sending/detecting DTMF (RFC 2833), DTMF grammar matching, and a handful of web requests, with total call duration of about 8 seconds - as the name says, a total performance stress test.

We believe we have the highest performance VoiceXML implementation in the industry, one particularly suited to the IVR industry.

Porting Your Applications to the SPOT SIP Engine

If you decide to port your VoiceXML IVR applications to SPOT SIP Engine, you will have the benefits that open standards bring, but you also will:

- Regain much of the performance of proprietary IVRs
- Have a choice in Speech Engines (via MRCP), if your application is speech enabled
- Be able to do more (calls per second) with substantially less resources (server hardware, operating systems licenses, floor space, power, operations personnel to manage a server farm)
- Enjoy a reduced Total Cost of Ownership

(To help in that decision, SPOT is available under Demo and Developer arrangements, in addition to Production licenses and Professional Services.)

And if your current IVR uses digital telephony for connecting to the outside world (SS7 or ISDN), there are a variety of Media Gateways supporting anywhere from a single T1/E1 trunk (24/30 channels/ports) all the way to high density gateway systems supporting as many as 32,000 channels. These are dedicated boxes for bi-directionally transforming between TDM calls (signaling and bearer channels) and VoIP calls (SIP signaling, RTP audio streams). As these Media Gateways become even more ubiquitous, IVRs with direct connections to the TDM world will be a thing of the past.

How easy is it to Port?

First, “application portability” isn’t truly real in the VoiceXML specification:

- Some VoiceXML features are optional - a platform may choose to implement them or ignore them
- Platform dependent features are explicitly built into the recommendation via:
 - session variables
 - properties
 - <object> tags

The VoiceXML Forum itself recognizes this problem. In the VoiceXML Review, Volume 7, Issue 1 - April/May 2007 article titled “The 2006 VoiceXML Forum Survey”, they state:

“Platform portability remains a weakness. The standard is not simple, and some features can be interpreted in various ways. Thus moving to a new voice platform can require a non-trivial amount of porting. Proprietary extensions are also an issue.”

A more appropriate analog might be that of porting an application from one Linux distribution to another – some are very simple, others are much harder to accomplish.

But that said, let’s set aside any issues due to custom objects, properties or session variables currently used in your application, when porting an existing VoiceXML application to SPOT, it is quite likely that the application will already fit within Interact’s conformant implementation, or any necessary changes to do so will be straightforward.

For specific details of what optional aspects of the standards we support, or don’t support, please see the SPOT SIP Engine Application Developer’s Reference.

Interact as your Technology Partner

Our interpreters in the SPOT SIP Engine are designed to function in a distributed (multi-host) and scalable environment - providing a pure software implementation for SIP based VoIP telephony.

As such, the SPOT SIP Engine differs significantly in certain design assumptions from those used by the designers of VoiceXML, so our interpreters' internals are very different from other interpreters' internals. It is those differences that yield our superior performance. These differences normally do not manifest themselves in VoiceXML applications or documents used in the typical way many VoiceXML documents are employed in the IVR world.

Our core design objectives are:

- Support creation of full featured voice applications via VoiceXML and CCXML documents
- Provide compliance with the World Wide Web Consortium's standards and the IETF, and ECMA specifications used within those standards
- Support a highly scalable, distributed server environment, with standby server redundancy
- Provide the highest performance possible within the constraints outlined above

To these, we can add additional design objectives and features oriented to the IVR and/or voice application developers:

- Bias for production environments and applications in large scale deployments,
- Deploy on the preferred IT operating system environment (Linux), with optionally available (and extensible) SNMP support
- Provide a web based status and management interface, Console, isolating the VoiceXML developer from having to be Linux specialist
- Support failover at the CCXML application layer, allowing for preserving established calls
- Provide application development support and related specifics in a reference guide suitable for professional IVR voice application developers, value added resellers and vertical market application specialists, and in-house enterprise specialists with responsibility for creating and maintaining their voice applications and their IVR's "voice user interface".
- Provide "carrier grade" support capability for integrators selling into the carrier market, with futureproofed SIP/VoIP capability mapped to CCXML events for answering, bridging, transferring, outbound dialing, teleconferencing, with easy integration of user-provided processes
- Provide hosted service providers easy tools to implement and deploy IVRs for their customers.

When we look at our capacity and performance in terms of channels and calls per second, not only have we met/exceeded our objectives, but are confident we run circles around any other commercially available voice engine.

The impact this has on TCO (Total Cost of Ownership) is significant. We can support more capacity and higher call rates on a given host than any other interpreter. Rather than throwing server hardware, operating system licenses, floor space, power, and people to manage a large server farm for a specific IVR application, we accomplish the same application with substantially less hardware, floor space, and money.

Take our SPOT SIP Engine Out for a Run!

You can do this in several ways:

- It is available on the www.iivip.com web site for free download and trial
- A SPOT Test Portal is available for playing with, and/or for creating/testing a voice application, available using a web browser and a SIP phone
- Our Professional Services team can give you a quote for porting your application, or porting a document and performance testing it for you. Call 1.800.242.8649 or +1.402.476.8786.
- A SPOT SIP Engine Development Kit ("SDK") is available allowing Developers to create and test commercial applications (but excludes running them in a production environment).
- Interact also has a portfolio of complete solutions available on the VIP Media Platform (using an earlier version of Interact's SPOT technology).

Further information is available at 1.800.242.8649, +1.402.476.8786 or via email yeswecan@iivip.com

About the Author

M. Wayne Wilson joined Interact Inc. Software Systems as Director of R&D in Austin in 2006. He has a PhD from Brown University in Applied Mathematics, spent 17 years with IBM, and has worked several years in a number of Austin area start-ups in a variety of management positions. His background includes mathematics, computer science, software development, videoconferencing and IVR development.