VoiceXML versus SALT: selecting a voice application standard

When it comes to speech application standards, it seems we've been asking all the wrong questions.

The VXML versus SALT debate is currently a hot topic in the IT conference rooms of organizations that rely on efficient, cost-effective contact centers. Phrases like ‘intense competition’ and ‘battle royale’ are bouncing around the trade press. Rival consortiums are at work writing specs and generating headlines, and some of the biggest names in technology have entered the VXML versus SALT fray.

So, which speech standard will win, VXML or SALT? A good question, but possibly a wrong question.

Given the real reasons organizations deploy speech-enabled technologies and the fundamental nature of technology standards themselves, our focus should be more on the application than on the application standard.

Ask yourself this: how often does someone using an accounting program or other application know or care whether that software was written in C or in Java-based code? When was the last time a customer hung up the phone and said, "Hey, that was the best VoiceXML application I have ever heard"?

The fact is, it's easy enough to get caught up in the debate over which standard is superior and which will dominate. Standards matter. But what matters most to the end user, and therefore what should matter most to contact centers and their system suppliers, is the quality and performance of the application itself.

With the end user in mind, now may be the time to ask some different and more relevant questions about VXML and SALT. The answers may surprise you.

The Future Of Speech

One thing is certain: speech recognition is the future of voice automation and a very important part of a customer or employee self-service strategy. In fact, some industry analyst firms indicate that over 80 percent of all customer interaction is still done with a telephone call.

Interactive voice response (IVR) is now a foundation technology of the customer service marketplace and is common in the contact centers of companies in financial services, insurance, telecommunications and a wide range of other industries.

As customer-oriented organizations seek to drive both service performance and cost efficiencies, the next major wave of investment is the incorporation of speech-based IVR solutions. The entire concept of IVR systems is being transformed by new and powerful speech-enabling innovations such as speech recognition, text-to-speech and speaker verification.

By incorporating speech technologies into their voice automation systems, enterprises are increasing productivity, cost efficiency and customer satisfaction.
But, as we have seen in so many other technology environments, the move to speech-driven automation has sparked an intense discussion over the relative merits and viability of the standards that underlie this still-emerging technology: Voice eXtensible Markup Language (VXML) and Speech Applications Language Tags (SALT).

These two evolving standards are making headlines as industry analysts, development groups and IT vendors jockey for position in the growing speech-enabled marketplace. Here are snapshot views of what their respective forums have to say about each.

**VoiceXML**

First published in 2000 by a consortium of 500 companies under the auspices of the VoiceXML Forum, VoiceXML has been described as the HTML of the voice Web. VXML is an open, standard markup language for voice applications. Originally developed for telephony applications, VXML harnesses the large Web infrastructure created for HTML to simplify the development and implementation of voice applications.

Control of the VoiceXML standard has been given to the Word Wide Web Consortium (W3C), and that group published the VoiceXML 2.0 version upon which a number of product solutions are now based. The VoiceXML Forum says VXML takes advantage of several industry trends, including the growth of the World Wide Web and the migration of the Web beyond the desktop computer, as well as improvements in computer-based speech recognition and text-to-speech synthesis.

**SALT**

As described by the SALT Forum, SALT extends existing Web markup languages such as HTML, XHTML and XML to enable multimodal and telephony access to the Web. The SALT 1.0 specification enables multimodal and telephony-enabled access to information, applications and Web services from personal computers, telephones, tablet PCs and wireless personal digital assistants.

This powerful multimodal access will allow end users to interact with applications in a number of ways, such as audio, speech and synthesized speech, plain text, mouse or keyboard, video or graphics. The SALT 1.0 specification is currently under consideration within the World Wide Web Consortium.

This is what the parties in each standard have to say about themselves. But how should managers of contact centers evaluate the relative pros and cons of SALT and VoiceXML?

**Five Questions To Ask**

If you are an IT manager responsible for the performance of a contact center, Web infrastructure or any other form of customer or employee self-service, and you see speech-enabled automation as a natural part of your user interface, which standard is right for you?

Here are five questions that go straight to the heart of the VXML versus SALT debate. Ask them, and get the right answers, before you make a decision on speech technology standards.

1. *What is your current Web infrastructure?*
Is your existing Web infrastructure built on J2EE or .NET? If you are developing applications in the .NET environment, the Microsoft Speech Server SALT browser provides a very clean, seamless integration. The Microsoft Speech SDK provides speech development tools and ASP.NET components that integrate into the Microsoft Visual Studio .NET development environment and the .NET application server. So for contact centers or Web development groups with an existing .NET Web environment, SALT is the obvious choice.

Companies that have adopted the J2EE Web infrastructure may have an easier time developing VoiceXML applications. Technically, VoiceXML and SALT browsers will work with any Web server. However, the development tools that are included by VoiceXML vendors are usually Java-based, while the tools included with the Microsoft SALT browser will obviously be tied to .NET. Java developers can still take advantage of the Microsoft Speech Server and development tools by using a .NET server with Web services that communicate with back-end J2EE components for data access and business transactions. While it is certainly possible to make SALT work in a Java environment, many J2EE-based organizations will probably choose VoiceXML.

2. Are speed and vendor support important to you?

If you want to deploy an open standards speech-enabled voice automation application rapidly on a proven technology platform, then VoiceXML provides an advantage in terms of time-to-market and a diversity of vendor offerings. Compared to the relatively new SALT standard, the more mature VoiceXML has been under development for several years and is now in its second major specification release. Additionally, product support for VoiceXML has been introduced by most (if not all) IVR vendors in the marketplace.

Organizations can leverage VoiceXML to immediately deploy an open-standards IVR with full integration to the call center, PBX, ACDs and CTI, and enjoy the technical support and service of established system suppliers. Vendors that support both the VoiceXML and SALT standards give companies an added degree of flexibility; they can deploy now using established VXML solutions and, if conditions warrant, migrate smoothly to SALT-based applications at some point in the future. In fact, a single customer or employee interaction could include interaction with both standards seamlessly within a single call.

3. Do you need multimodal access?

If multimodal access by devices including mobile phones and wireless PDAs, in addition to traditional telephony and Web browsers, will be an important consideration in the future, then the SALT standard makes better sense than VoiceXML. Multimodal access is a core capability of the SALT specification, while VoiceXML is a voice interface language originally designed specifically for the voice user interface.

This doesn't mean you can't voice-enable a Web site using VoiceXML. X+V (or XHTML plus Voice) extends the VoiceXML specification by adding multimodal attributes. However, if multimodal access is a central issue in your deployment, SALT may be your best option given its more granular level control of multimodal events and the fact that this capability was built into the requirements of its design from day one.
For example, the New York City Department of Education (NYC DOE) boasts the largest school system in the country with over one million students. To optimize the children's educational experience by addressing parental concerns and encouraging parental involvement, the NYC DOE is using a speech-enabled application on a SALT-powered platform to enable parents to check things such as their child's attendance record, course grades and lunch menu for the day. Much of this information is already available to the parents via the NYC DOE Web site, but they are using speech technologies to enable round-the-clock accessibility to the information for parents that don't have consistent access to a computer.

4. **Which standard will best support your existing infrastructure?**

The standard you select must support your existing technology infrastructure. However, the plain truth is that both VoiceXML and SALT are equally inadequate in their ability to integrate into a call center environment. VoiceXML and SALT are presentation-layer specifications, meaning they address the user interface (voice and multimodal), but do not address integration or back-end functionality requirements.

At best, standards provide a baseline framework for such things as the hardware platforms (Intel, Windows, Linux), telephony integration (ISDN, SS7, SIP) and the voice user interface (VoiceXML and SALT). But standards do not encompass all of the components needed to integrate and deploy a voice automation solution. If we consider technologies such as call control, CTI and legacy host integration, we see that standards do not cover every crucial element necessary for a successful voice automation solution. Components that also include tools for operating, maintaining and administering the systems and tools for developing and debugging applications are needed to support the successful lifecycle of a solutions deployment and are not adequately addressed by the standards themselves.

To create a workable solution, you need all of these elements - some of which are supported by an open standard. However, in the end, it's up to the solutions vendor to provide all of the product components that are needed to develop, maintain and report on a voice application. In fact, an organization can rely on open standards to build, perhaps, half of an open IVR solution, and the rest must be supplied by a vendor. Because neither SALT nor VXML provide all of the features needed for an IVR solution, organizations that must deploy these solutions in the context of the larger call center environment may wish to seek out solutions that support both SALT and VXML.

There are no agreed upon standards for call control, CTI or data/host integration, for example, which means various vendors will deploy very unique solutions. To ensure optimum flexibility, it is important to support either your preferred standard or both standards, so you can support the elements of your existing or planned infrastructure under that standard.

5. **Which standard will win in the contact center market?**

The answer is, we just don't know and in reality we don't need to know. There will always be new and competing ideas on standards, and while either SALT or VXML may one day emerge as the dominant player, they may coexist equally for a significant period of time. In fact, there's even talk that the two standards may one day come together into one. The other thing of which we can be certain is that standards such as VXML and SALT will continue to evolve, and that new standards will
be created to address new functionality in the future. That's why it makes no sense to delay the
launch of a flexible, cost-efficient voice automation solution until the standards sort themselves out.

Standards In Perspective

It is easy enough to get caught up in the debate over the relative advantages of one standard or
another. Standards matter. What matters more is that you get the speech-enabled application right.
In the end, customers interface with and react to applications and not the standards that help enable
those applications.

By keeping your focus on the quality and efficiency of the customer interaction, and the wider set of
automated voice technologies needed to support more natural and effective communications, you
can put the ongoing debate over standards in its proper perspective.

Organizations invest in voice recognition solutions to improve the quality of customer interactions.
Standards are just a means to that more important end.

Bron:
http://www.thefreelibrary.com/Voicexml+versus+salt:+selecting+a+voice+application+standard-
a0117923538, geraadpleegd op donderdag 7 februari 2008
VoiceXML CCXML SALT

There's been an industry shift from using proprietary approaches for developing speech-enabled applications to using strategies and architectures based on industry standards. The latter offer developers of speech software a number of advantages, such as application portability and the ability to leverage existing Web infrastructure, promote speech vendor interoperability, increase developer productivity (knowledge of speech vendor's low-level API and resource management is not required), and easily accommodate, for example, multimodal applications. Multimodal applications can overcome some of the limitations of a single mode application (GUI or voice), thereby enhancing a user's experience by allowing the user to interact using multiple modes (speech, pen, keyboard, etc.) in a session, depending on the user's context.

VoiceXML, Call Control eXtensible Markup Language (CCXML), and Speech Application Language Tags (SALT) are emerging XML specifications from standards bodies and industry consortia that are directed at supporting telephony and speech-enabled applications. The purpose of this article is to present an overview of VoiceXML, CCXML, and SALT and their architectural roles in developing telephony as well as speech-enabled and multimodal applications.

Before I discuss VoiceXML, CCXML, and SALT in detail, let's consider a possible architectural deployment that employs these specifications. At a high level are two main architectural components: document server and speech/telephony platform. Each interfaces with a number of secondary servers (Automated Speech Recognition server (ASR), Text-to-Speech server (TTS), data stores).

In this architecture a document server generates the documents in response to requests from the speech/telephony platform. The document server leverages a Web application infrastructure to interface with back-end data stores (message stores, user profile databases, content servers) to generate VoiceXML, CCXML, and SALT documents. Typically, the overall Web application infrastructure separates the core service logic (the business logic) from the presentation details (VoiceXML, CCXML, SALT, HTML, WML) to provide a more extensible application architecture. The application infrastructure is also responsible for maintaining application dialog state in a form that's separate from a particular presentation language mechanism.

To process incoming calls, the speech/telephony platform requests documents from the document server using HTTP. A VoiceXML or CCXML browser that resides on the platform interprets the VoiceXML and CCXML documents to interact with users on a phone. Typically, the platform interfaces with the PSTN (Public Switched Telephone Network) and media servers (ASR, TTS) and provides VoIP (SIP, H.323) support. An ASR server accepts speech input from the user, uses a grammar to recognize words from the user's speech, and generates a textual equivalent that is used by the platform to decide the next action to take, depending on the script. A TTS server accepts markup text and generates synthesized speech for presentation to a user. In this deployment a SALT browser on a mobile device interprets SALT documents. Figure 1 is a diagram illustrating such an architecture.
VoiceXML

Now that you have an overall understanding of the architecture in which these specifications can be used, let's begin by discussing VoiceXML. VoiceXML can be viewed as another presentation language (HTML, WML) in your architecture. VoiceXML is a dialog-based XML language that leverages the Web development paradigm for developing interactive voice applications for devices such as phones and cell phones. It's a self-contained presentation language designed to accept user input in the form of DTMF (touch tones produced by a phone) and speech, and to generate user output in the form of synthesized speech and prerecorded audio. It isn't designed to be embedded in an existing Web language (e.g., HTML, WML) and leverage HTML's event mechanism. At this writing, VoiceXML 2.0 is a W3C working draft (www.w3.org/TR/voicexml20/). VoiceXML is currently used to support a number of different types of solutions - for example, to automate the customer care process for call centers and to support sales force automation to enable a sales agent to access appointments, customer information, and address books by phone. It's also used in unified communications solutions to enable a user to manage his messages (e-mail, voice, fax), including personal information (personal address books).

To appreciate how VoiceXML can be used, it's necessary to understand its structure, elements, and mechanisms. A VoiceXML application comprises a set of documents sharing the same application root document. When any document in the application is loaded, the root document is also loaded. Users are always in a dialog defined in a VoiceXML document (i.e., the interaction between a user and voice/telephony platform is represented in a dialog). Users provide input (DTMF or speech) and then, based on application logic in the document, are shown a new dialog that presents output (audio files or synthesized speech) and accepts further user actions that can result in a new dialog. Execution ends when no further dialog is defined. Transitions between dialogs use URIs.

There are two main types of dialogs: forms and menus. Forms pre-sent output and collect input while menus present a user with choices. Fields are the building blocks of forms and comprise prompts, grammars (describing allowable inputs), and event handlers. VoiceXML's built-in Form Interpretation Algorithm (FIA) determines the control flow of a form. For each form element the main FIA routine involves selecting a form item, collecting input (playing a prompt, activating grammars, waiting for input or event), and then processing the input or event. Examples of actions defined in a form include collecting user input (<field>), playing a prompt (<prompt>), or executing an action when an input variable is filled (<filled>). The FIA's interpretation of a form ends when an <exit> element or transition to another dialog (<submit>, <goto>) is encountered.

Menus (<menu>) can be loosely described as a special case of a form with a single, implicitly defined field composed of a set of choices (<choice>). A choice element (<choice>) defines a grammar that determines its selection and a URI to transition to. Menus can be speech, DTMF, or both.

VoiceXML supports both system-directed (for less experienced users) and mixed-initiative conversations (for more experienced users); <link> is used to support mixed-initiative dialogs. VoiceXML defines specific elements to control dialog flow and handle user interface events. Elements such as <var>, <if>, and <goto> are provided to define application logic, and
ECMAScript code can be defined in <script> elements. VoiceXML offers elements to handle situations when there's no user input or the input isn't understandable through its event mechanism (<throw>, <catch>, <noinput>, <error>, <help>, <nomatch>). These and other VoiceXML elements are used in Listing 1 to illustrate how VoiceXML could be used to support a simple-content services application (voice portal for getting stock quotes, news, weather, etc.).

VoiceXML's grammar activation is based on the scope in which the grammar was declared and the current scope of the VoiceXML interpreter. For example, declaring a grammar in the root document means that the grammar will be active throughout the execution of the VoiceXML application. Grammars can be active within a particular document, form, field, or menu, and can be inline, external, or built in (e.g., Boolean, date, phone, time, currency, digits). Although, unlike VoiceXML 1.0, it doesn't preclude other grammar formats, version 2.0 requires support for the XML form of the Speech Recognition Grammar Specification (SRGS). VoiceXML 2.0 interpreters should also be able to support Speech Synthesis Markup Language (SSML) specification for synthesized speech. As I write, both SSML and SRGS are W3C working drafts.

VoiceXML has some other features to promote architectural extensibility and reusability. For example, the <object> element offers facilities for leveraging platform-specific functionality while still maintaining interoperability, and the <subdialog> element can be used to develop reusable speech components.

While it provides a suitable language for developing interactive speech applications, VoiceXML lacks support for features such as conferencing, call control, and the ability to route and accept or reject incoming calls. <transfer>, its tag for transferring calls, is inadequate for these types of features. Further, VoiceXML's execution model isn't well suited to an environment that needs to handle asynchronous events external to the VoiceXML application. VoiceXML can handle synchronous events only - those that occur only when the application is in a certain state.

**CCXML**

CCXML addresses some of the telephony/call control limitations of VoiceXML. It enables processing of asynchronous events (events generated from outside the user interface), filtering and routing of incoming calls, and placing outbound calls. It supports multiparty conferencing as well as the creation/termination of currently executing VoiceXML instances and the creation of a VoiceXML instance for each call leg. A CCXML browser on a voice/telephony platform interprets CCXML documents. CCXML is currently a W3C working draft (www.w3.org/TR/ccml/).

Since CCXML is still an emerging specification, few deployed solutions are in the market today. However, it can be used to support a number of different types of applications. Conferencing applications can be developed using it. Voice messaging applications can use it to enable a user to filter incoming calls and route them to a particular application. CCXML can also be used to support notification functions (e.g., place outbound calls to notify a user of a new appointment or a stock alert).
The structure of a CCXML program reflects its fundamental value in being able to handle asynchronous events. Processing of events from external components, VoiceXML instances, and other CCXML instances is central to CCXML. A CCXML program basically consists of a set of event handlers () for processing events in the CCXML event queue. Each element comprises elements. An implicit Event Handler Interpretation Algorithm (EHIA) interprets <event-handler> elements. A CCXML interpreter essentially removes an event from its event queue, processes it (selects <transition> elements in <event-handler> that match the event and performs actions within that <transition> element), and then removes the next item from the event queue.

Within a <transition> element a new VoiceXML dialog can be started (<dialogstart>) and associated with a call leg. Note that the dialog launch is nonblocking, and control is immediately returned to the CCXML script. A CCXML script can also end a VoiceXML dialog instance (<dialogterminate>). Conditional logic (, ) can be used in a <transition> element to accept or reject incoming calls or modify control flow. Incoming calls can be accepted or rejected (<accept>, <reject>), and outbound calls can be placed (<createcall>). A CCXML application can contain multiple documents, and a CCXML execution flow can transition from one document to another (<goto>, <fetch>, <submit>) and end its execution (<exit>). A CCXML instance can also create another CCXML instance (<createccxml>) with a separate execution context and send events (<send>) to other CCXML instances. Multiparty conferences can be created and terminated (<createconference>, <destroyconference>), and call legs can be added to and removed from a conference (<join>, <unjoin>).

For telephony events the JavaSoft call model is used to provide the abstractions (Address, Call, Connection, Provider). A Call object is a representation of a telephone call. A call comprises zero or more Connections (e.g., a conference call typically has three or more connections). A Connection describes the relationship between a Call and an Address (e.g., telephone number). A Connection is in one of a defined set of states at a particular time. For example, when a Connection and an Address are an active part of a call, a specific event is emitted (connection.Connection_CONNECTED), whereas when an address is being notified of an incoming call, a different event is emitted (connection/connection_ALERTING).

The code snippet in Listing 2 illustrates the main CCXML elements involved in processing an incoming call notification event and then processing a subsequent connected event. Based on a connection.

CONNECTION_ALERTING event, the call could be accepted or rejected according to the called ID using conditional logic (if, else). When a connection enters the connected state, a VoiceXML dialog (login.xml) is launched to perform some authentication (e.g., entering a PIN number). Listing 2 also illustrates the main structure of a CCXML script to address this scenario.

Other types of events processed by a CCXML script are sent from VoiceXML instances or from other CCXML instances. For example, a CCXML script can capture status from a terminating VoiceXML dialog because VoiceXML's <exit> element allows variables to be returned to the CCXML script. When a VoiceXML interpreter ends execution, dialog.exit event is received by a CCXML instance. Using a <transition> element, the CCXML script can process the dialog.exit event and access variables returned by the VoiceXML dialog. For
example, if we have a VoiceXML dialog that presents a set of menu options, upon termination of the VoiceXML dialog the CCXML script can obtain the selected menu choice and act on it (e.g., perform an outdial, join a conference). Events are also used to communicate between CCXML instances. However, CCXML currently doesn’t define a specific transport protocol for communication; SIP and HTTP are possibilities for the underlying transport.

A number of factors currently inhibit the adoption of CCXML.

- Since it’s a new specification still under review, there are few CCXML browser implementations compared to VoiceXML browser implementations. Note that although it’s designed to complement a dialog-based language such as VoiceXML, a CCXML system isn’t required to support a VoiceXML implementation. If CCXML and VoiceXML are used together, the instances would run separately.
- While CCXML is a promising start in addressing some of the limitations of VoiceXML, a number of areas remain to be specified in order to meet the needs of call control applications. For example, while CCXML is and should remain uncoupled from a specific underlying protocol, protocol-agnostic mechanisms could be specified to allow passing in protocol-specific parameters (VoIP, SS7) when performing certain functions (e.g., making an outbound call).
- Other items for future discussion are documented in the current CCXML specification (e.g., communication between different CCXML instances).

**SALT**

SALT enables speech interfaces to be added to existing presentation languages (e.g., HTML, XHTML, WML) and supports multimodal applications. PDAs, PCs, and phones are examples of devices that can support SALT applications. At the time of this writing, the SALT Forum (www.saltforum.org), an industry consortium, has published a 0.9 version of the SALT specification.

Since SALT is an emerging specification, at the time of this writing there are no known deployed solutions. However, SALT can enhance a user’s experience on PDAs and other mobile devices that have inherent limitations, such as keyboards that are difficult to use and small visual displays. Multimodal solutions allow a user to use multiple I/O modes concurrently in a single session. Input can be via speech, keyboard, pen, or mouse, and output can be via speech, graphical display, or text. For example, a multimodal application can enable a user to enter input via speech and receive output in the form of a graphic image. Thus a user can select the appropriate interaction mode depending on the type of activity and context. SALT can enable solutions that enhance a user’s experience when using applications such as content services (news, stock quotes, weather, horoscope), unified communications, sales force automation, and call center support.

The core philosophy behind the SALT specification is to use a lightweight approach to speech-enabling applications. This is evident in the small set of tags in the SALT specification that can be embedded in a markup language such as HTML for playing and recording audio, specifying speech synthesis configuration, and recognizing speech. The \(<\text{listen}>\) element is
used to recognize input and define grammars (<grammar>). SALT browsers must support the XML form of the W3C's SRGS.

This means that the XML form of SRGS can be used in SALT's <grammar> element. The <listen> element also has an element for processing the input (<bind>). The <bind> element processes the input result that's in the form of a semantic markup language document. A SALT browser must support W3C's Natural Language Semantic Markup Language (NLSML) for specifying recognition results. XPath is used to reference specific values in the returned NLSML document that are then assigned to a target defined in the . Thus input text can be obtained from either a graphical display or, now, using a speech interface.

The <listen> element can also contain a single <record> element for capturing audio input. SALT also supports DTMF input using the <dtmf> element; like <listen>, its main elements are <grammar> and <bind>. For audio output the <prompt> element can represent a prompt using a speech synthesis language format (SALT browsers must support SSML). This means that SSML can be embedded within SALT's <prompt> elements.

SALT's call control functionality is provided in the form of a call control object model based on the JCP Call Model and portions of Java Telephony API (JTAPI). The callControl object provides an entry point to the call control abstractions and consists of one or more provider objects. Providers represent an abstraction of a telephony protocol stack (SIP, H.323, SS7). Providers create and manage a conference. An address is a communication endpoint (e.g., SIP URL for VoIP), and providers define the available addresses that can be used. A conference is composed of one or more calls that can share media streams. SALT's call control objects define states, transitions, events, and methods for supporting call control. For example, there are methods for beginning a new thread of execution for processing an incoming call (spawn() method of the callControl object), answering an incoming call (accept() method of call object) based on an alerting event (call.alerting), or transferring a call (transfer() method of call object).

Also, SALT has a <smex> element that can be used to exchange messages with an external component of the SALT platform. Functionality such as Call Detail Record (CDR) generation, and logging or proprietary features, can be leveraged using this extensible mechanism while maintaining interoperability. For example, <smex> can be used to leverage call control functionality on a different platform (e.g., send call control requests to and receive events from a CCXML interpreter on a remote host). In this situation SALT can be used for dialog management while call control is handled by a separate architectural entity (e.g., CCXML interpreter) on a remote host.

Let's consider how SALT elements can be embedded in an existing Web page to also offer a speech interface. Assume a Web page with SALT elements allows a user to select the latest news for a particular sport (soccer, basketball, football, tennis) by selecting the sport from a drop-down box. To enable speaking the name of the sport, the start() method of listen's DOM object is called to invoke the recognition process. The value returned from the recognition is assigned to an input field (e.g., txtBoxSport). As with other SALT DOM elements, the listen element also defines a number of events (e.g., onreco, onsilence, onspeechdetected) whose handlers may be specified as attributes of the listen object (i.e.,
event-driven activation is an inherent feature of SALT). The following code snippet illustrates some of the SALT elements involved in this scenario:

```xml
...<input name="txtBoxSport" type="text" onpendown="listenSport.start()"/>
<salt:listen id="listenSport">
  <salt:grammar name="gramSport" src="/sport.xml"/>
  <salt:bind targetElement="txtBoxSport" value="//sport"/>
</salt:listen>
...```

You've probably noted that SALT and VoiceXML can be used to develop dialog-based speech applications, but the two specifications have significant differences in how they deliver speech interfaces. Whereas VoiceXML has a built-in control flow algorithm, SALT doesn't. Further, SALT defines a smaller set of elements compared to VoiceXML. While developing and maintaining speech applications in two languages may be feasible, it's preferable for the industry to work toward a single language for developing speech-enabled interfaces as well as multimodal applications.

**Summary and Conclusion**

This short discussion has provided a brief introduction to VoiceXML, CCXML, and SALT for supporting speech-enabled interactive applications, call control, and multimodal applications and their important role in developing flexible and extensible standards-compliant architectures. This presentation of their main capabilities and limitations should help you determine the types of applications for which they could be used.

The various languages expose speech application technology to a broader range of developers and foster more rapid development because they allow for the creation of applications without the need for expertise in a specific speech/telephony platform or media server. The three XML specifications offer application developers document portability in the sense that a VoiceXML, CCXML, or SALT document can be run on a different platform as long as the platform supports a compliant browser.

These XML specifications are posing an exciting challenge for developers to create useful, usable, and portable speech-enabled applications that leverage the ubiquitous Web infrastructure.

**References**

For more information on using these XML specifications in your speech-based system architectures, see the following:

- Voice browser call control (CCXML v. 1.0): [www.w3.org/TR/ccxml](http://www.w3.org/TR/ccxml)
- Speech Application Language Tags 0.9 Specification (draft): [www.saltforum.org](http://www.saltforum.org)
- Voice Extensible Markup Language (VoiceXML v. 2.0): [www.w3.org/TR/voicexml20](http://www.w3.org/TR/voicexml20)
- Speech Synthesis Markup Language Specification: [www.w3.org/TR/speech-synthesis](http://www.w3.org/TR/speech-synthesis)
• Speech Recognition Grammar Specification for W3C speech interface framework: www.w3.org/TR/speech-grammar
• Natural Language Semantics Markup Language for speech interface framework: www.w3.org/TR/nl-spec

Bron: http://xml.sys-con.com/read/40476.htm, geraadpleegd op donderdag 7 februari 2008