

Text-to-Speech Markup Languages

Andrew Lampert

October 2004

Introduction.....	2
The Evolution of Speech Synthesis Markup Standards.....	3
Prosodic Category Notation System.....	3
(The original) Speech Synthesis Markup Language (SSML).....	4
Spoken Text Markup Language (STML).....	4
Java Speech API Markup Language (JSML).....	5
SABLE.....	7
W3C Speech Synthesis Markup Language (SSML).....	10
Apple Speech Synthesis Manager.....	13
Microsoft Speech API (SAPI).....	14
Microsoft Speech Application Software Development Kit (SASDK).....	16
VoiceXML (VXML).....	17
Speech Application Language Tags (SALT).....	19
References.....	20

Introduction

Plain text alone does not contain enough information for a speech engine to know how to accurately synthesize it with speech. There are many ambiguous and implicit features of text that cannot accurately be inferred from text alone. Examples of such ambiguities include word pronunciations and paragraph delineation. Similarly, the prosody and intonation of synthesized speech cannot be inferred with complete accuracy without metadata accompanying the text. Other speaker directives, such as speaker/voice characteristics, are also absent in plain text.

Over many years of research into and commercial development of speech synthesis systems, a large number of competing speech markup standards have been proposed. Many of these have been proprietary standards, specific to a particular vendor's speech engine or platform.

A number of attempts have been made to create open standards for speech markup, and ultimately to develop a common markup control language for text-to-speech synthesis. It should be noted, however, that not all these languages have the same goals.

Many of the speech markup languages explored in this document aim to provide the ability for experts and non-experts to markup input text to guide speech synthesis. Examples of such languages include SSML (both the W3C and original Edinburgh versions), STML, JSML, SABLE and SAPI. While there are syntactic differences between these languages, they are functionally similar, and are all intended to be independent of any particular TTS system. Each specific TTS engine needs to parse and translate the markup annotations into system-specific representations to control the synthesis process.

There are also other classes of speech markup languages. Some belong to a class of proprietary and system specific markup languages, such as Kohler's input markup for the RULSYS/INFOVOX TTS system, and Bell Lab's proprietary set of escape sequences for their own TTS system.

Others are much wider in scope than synthesis alone. Examples include VoiceXML (along with its predecessors - Motorola's VoxML and IBM's SpeechML) and SALT which combine parts of speech synthesis markup with speech recognition, dialogue management, and telephony specific functionality.

The Evolution of Speech Synthesis Markup Standards

Prosodic Category Notation System

Klaus Kohler is a professor in the Institute for Phonetics and digital Speech Processing at the Christian-Albrechts-Universität, Kiel. In 1994, Kohler wrote a paper [9] outlining the Kiel Intonation Model (KIM) for German, incorporating stress and intonation, timing and articulatory reduction. The categories of this model are represented using a *prosodic notation system*. This prosodic notation is symbolized with a set of ASCII character annotations which are made to speech data in order to control prosody of synthesized speech.

The focus of this work was certainly not on creating an open standard for speech markup, instead, what is discussed is a proprietary language for controlling prosody in synthesized speech for the RULSYS/INFOVOX German TTS System at Kiel. It is, however, one of the earliest know examples of a published speech markup specification.

A table of some of the ASCII characters used, and the corresponding prosodic categories they represent, is shown in Table 1.

Table 1: ASCII Characters representing Prosodic Categories in [Kohler, 1994]

ASCII Character	Prosodic Category
Apostrophe [']	Primary stress vowels
Quotation Mark [“]	Secondary stress vowels
Hash [#]	Phonetically relevant word boundary
3-Digit [3]	Word Reinforcement
2-Digit [2]	Word accent
1-Digit [1]	Partially de-accented sentence stress
0-Digit [0]	Completely de-accented sentence stress
Plus [+]	Function words
Period [.]	Pitch peak
Comma [,]	Low rising valley
Question Mark [?]	High rising valley
Close Parenthesis [)]	Early peak position in sentence stress
Open Parenthesis [(]	Late peak position in sentence stress
P-Colon [p:]	Prosodic boundary marker. Preceded by two digits which specify utterance-final lengthening, and pause length respectively. For pitch peaks, a third digit refers to the scaling of the F0 endpoint.

Digit [0,1,2,3], preceding P-Colon	<p>Speech rate, including degrees of reduction or elaboration.</p> <ul style="list-style-type: none"> • 0 – Degrees of reduction and speed are increased. • 1- Medium speed and higher degree of reduction. • 2 – Medium overall speed and default reduction. • 3 – Decreased reduction and speed.
------------------------------------	--

Since 1997, the syntax for speech markup has been augmented as follows:

- All ASCII prosodic labels are now prefixed with an ampersand [&]. This makes it possible to keep orthographic punctuation,
- Disfluencies (break-offs and resumptions) are marked by [+ / +], or by [+ / = / +] inside a word.
- Breathing pauses are marked by [h:]

(The original) Speech Synthesis Markup Language (SSML)

Probably the first attempt at creating an open standard for speech markup was the Speech Synthesis Markup Language (SSML), not to be confused with the much more recent W3C standard of the same name. This original SSML [15] was the name of a markup language developed as part of a PhD thesis [8] done at the University of Edinburgh in 1995. Version 1.0 of the Edinburgh SSML was based on SGML, and consisted of only four elements which offered the ability to:

- Mark phrase boundaries,
- Emphasise words,
- Specify pronunciations, and
- Include other sound files.

Proposed additions for version 2.0 of Edinburgh SSML included ‘style files’, which would allow authors to specify speaker/voice specific parameters for segments of text, and a standard for mathematical markup, which was intended to be based on the system used in HTML 3.0 [13].

Spoken Text Markup Language (STML)

Building on the work done for SSML, Paul Taylor and Amy Isard (the original SSML authors) from Edinburgh University and Richard Sproat and Michael Tanenblatt from Bell Labs worked collaboratively to create a specification for the Spoken Text Markup Language (STML) [15]. STML was a successor to the markup languages used by both Edinburgh and Bell Labs, that is, it superseded both Edinburgh’s SSML and Bell Labs’

proprietary set of escape sequences [6] which were used to control their respective synthesis engines. The aim for STML was again to make a markup language that worked with a variety of synthesis engines. In doing so, the synthesized speech was not intended to sound identical on different systems, rather the focus was on giving the same basic impression to the listener.

In both generality and coverage, STML was claimed to be (at the time), superior to “the only extant industry standard for TTS markup, namely Microsoft’s Speech Application Programmer’s Interface (SAPI)” [15]. Like SSML, STML was based on SGML, but it defined a more extensive set of tags than SSML. Additions included tags for specifying speaker and language, as well as tags to specify the genre of the text to be synthesized. Example genres proposed included plain prose, poetry and lists.

Java Speech API Markup Language (JSML)

Around 1997, Sun Microsystems finished development of its own speech markup language specification, the Java Speech API Markup Language (JSML) [18], which it released as part of its Java Speech API (JSAPI). JSAPI defines an abstract Application Programming Interface (API) in the form of a set of abstract classes and interfaces that represent a Java programmer’s view of a speech engine. JSAPI also defined JSML and the Java Speech API Grammar Format (JSGF) as companion specifications.

Sun publicly released version 1.0 of JSAPI in 1998. In developing version 1.0 of JSML, Sun acknowledged contributions from Apple Computer, AT&T, Dragon Systems, IBM, Novell, Philips Speech Processing and Texas Instruments. Sun also explicitly acknowledged the heritage of JSML, stating that it had benefited from previous SGML and XML based markup initiatives, namely SSML (Edinburgh), SABLE, and work at MIT [17].

Later, in June 2000, the W3C published a note about JSML, entitled “JSpeech Markup Language” [7]. This note was published based on a submission from Sun, and contains the same technical information as Sun’s JSML specification. The change in name (to JSpeech Markup Language) was purely to protect Sun’s trademarks.

The Java Speech API Markup language (JSML) is an XML [1] based markup language, which allows text to be annotated with additional metadata to guide synthesized pronunciation. In the same spirit as the Java language itself, one of the primary aims of JSML was to create a consistent API for speech applications which would enable applications to be portable across different synthesizers and computing platforms.

As an XML based markup language, JSML annotations are tag based. The JSML element set includes several types of element. First, JSML documents can include structural elements that mark paragraphs and sentences. Second, there are JSML elements to control the production of synthesized speech, including the pronunciation of words and phrases, the emphasis of words (stressing or accenting), the placement of boundaries and

pauses, and the control of pitch and speaking rate. Finally, JSML includes elements that represent markers embedded in text and that enable synthesizer-specific controls.

The example in Figure 1 shows JSML markup for the body of an email. Tags for annotating both text structure and pronunciation hints are illustrated.

```
<jsml>

<div type="paragraph">Message from
<emphasis>Alan Schwarz</emphasis> about new synthesis
technology. Arrived at <sayas class="time">2pm</sayas>
today.</div>

<div type="paragraph">I've attached a diagram showing the new way we do
speech synthesis.</div>

<div>Regards, Alan.</div>

</jsml>
```

Figure 1: An example of JSML Markup (extracted from [18])

The set of tags available in the current JSML specification (v 0.6) [18] is shown in Table 2 below.

Table 2: JSML Markup Tags

Element Function	Element Name	Element Type	Element Description
Structure	jsml	Container	Root element for JSML documents.
	div	Container	Marks text content structures such as paragraph and sentences.
Production	voice	Container	Specifies a speaking voice for contained text.
	sayas	Container	Specifies how to say the contained text.
	phoneme	Container	Specifies that the contained text is a phoneme string.
	emphasis	Container	Specifies emphasis for the contained text or immediately following text.
	break	Empty	Specifies a break in the speech.
	prosody	Container	Specifies a prosodic property, such as baseline pitch, rate, or volume, for the contained text.

Miscellaneous	marker	Empty	Requests a notification when speech reaches the marker.
	engine	Container	Native instructions to a specified speech synthesizer.

There are several limitations in the JSML specification. One such limitation is that lack of explicit downstep and declination tags. These can, however, be implemented by appropriately resetting the *pitch* and *range* values of the *emphasise* tag.

SABLE

After their work on STML, Edinburgh and Bell Labs continued to collaborate and work towards developing a single speech markup standard. First published in 1998, the SABLE specification [16] was coauthored by researchers from the University of Edinburgh and Bell Labs, along with representatives from Sun Microsystems, Boston University, CMU and BT Labs.

SABLE was designed to provide a single standard for speech synthesis markup. The principal aim was to address the problems created by the existence of a large number of incompatible and proprietary speech synthesis control languages. To address these issues, the authors based the Sable standard on both STML and JSML.

The stated aims for SABLE were:

- **Synthesizer Control.** SABLE should enable markup of TTS text input for improving the quality and appropriateness of speech output.
- **Multilinguality.** SABLE should be suitable for marking up text for synthesis in any language.
- **Ease of Use.** The SABLE markup must be easy to use, such that users should not require specialized knowledge of TTS or linguistics to annotate text with SABLE markup.
- **Portability.** SABLE should provide application developers with a consistent syntax and mechanism for controlling heterogeneous synthesizers across multiple platforms.
- **Extensibility.** SABLE should be flexible enough to support new features in future releases. Additionally, to encourage research, individual synthesizers should be able to define additional features without compromising the portability of SABLE text.

Like the JSML and STML standards from which it was derived, SABLE supported markup for two purposes. The first type of markup annotates the properties of text structure that are relevant for rendering a document in speech. In SABLE, text structure is mostly represented using the DIV tag, whose TYPE attribute can be set to values such as *sentence*, *paragraph* or *stanza*. The SAYAS tag also annotates the text structure;

specifically it identifies the function of the contained region of text (eg. as a date, an email address, a currency figure etc). These text structure tags are derived from the *text description* tags in STML and the *structural elements* in JSML.

The second type of markup provides control over the speech production. Examples of such tags include EMPH, which marks emphasis, PITCH, which specifies voice frequency, RATE, which controls the speed of speech, and PRON, which provides phonemic specification for pronunciation using the International Phonetic Alphabet (IPA). In JSML and STML, this second type of markup tag was known as either a *production element* or a *speaker directive*.

As with JSML, one of SABLE limitation is that lack of explicit downstep and declination tags. Like JSML, however, these can be implemented by appropriately resetting the *pitch* and *range* values of the *emphasise* tag.

Like JSML, SABLE also specified a MARK attribute, which sets an arbitrary marker. This provides a callback mechanism for applications to be notified when specific parts of the SABLE text have been processed.

The full set of SABLE tags is shown below in Table 3.

Table 3: SABLE Markup Tags

Element Function	Element Name	Element Type	Element Description
Text Description	sayas	Container	Define the mode in which contained text is to be said. Examples include date, time, phone, currency etc.
	div	Container	Marks the contained region as structural text of a specified type. Example types include sentence, paragraph etc.
Speaker Directives	speaker	Container	Specifies properties of the speaking voice for contained text. Eg. age, gender, name.
	language	Container	Specifies the language of the contained text.
	sable	Container	Root element for SABLE documents.
	pron	Container	Substitute the specified pronunciation for what would normally correspond to the contained text. Pronunciation can be defined using IPA or “phonetic” spelling in the language of the enclosing text.

	emph	Container	Specifies emphasis for the contained text.
	break	Empty	Specifies an intrasentential, prosodic break in the speech at the current position. The level, length and type can be specified.
	pitch	Container	Specifies a the pitch properties (base, middle range) of the enclosed text, using either numeric or descriptive values.
	rate	Container	Sets the average speech rate of the enclosed region, using either numeric or descriptive values.
	volume	Container	Sets the amplitude of the enclosed region, using either numeric or descriptive values.
	audio	Container	Load and play an audio URL.
	marker	Empty	Requests a notification when speech reaches the marker.
	engine	Container	Native instructions to a specified speech synthesizer.

W3C Speech Synthesis Markup Language (SSML)

The W3C's Speech Synthesis Markup Language (SSML) [4] is based largely on the Java Speech API Markup Language (JSML), but also takes inspiration and concepts from the SABLE specification. SSML also adds new elements, not present in either of the JSML or SABLE standards.

The aim of SSML is to provide a feature rich, XML-based markup language for controlling and guiding the generation of synthesized speech in Web and other applications. As with SABLE, JSML etc., the main role of SSML is to provide a standard way to control aspects of speech such as pronunciation, volume, pitch, and rate across multiple heterogeneous speech synthesis platforms.

The example shown below in Figure 2 illustrates the reading of email headers. The p and s elements are used to mark text structure. The break element is placed before the time to signify that the time is important information for this application. The prosody element is used to slow the speaking rate of the email.

```
<?xml version="1.0"?>
<!DOCTYPE speak PUBLIC "-//W3C//DTD SYNTHESIS 1.0//EN"
    "http://www.w3.org/TR/speech-
synthesis/synthesis.dtd">
<speak version="1.0"
    xmlns="http://www.w3.org/2001/10/synthesis"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
    http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
    xml:lang="en-US">
  <p>
    <s>You have 4 new messages.</s>
    <s>The first is from Stephanie Williams and arrived at <break/>
3:45pm.
    </s>
    <s>
      The subject is <prosody rate="-20%">ski trip</prosody>
    </s>
  </p>
</speak>
```

Figure 2: An example of SSML markup (extracted from [4])

While SSML is based upon the JSML specification, the work done in SABLE to combine multiple XML and SGML based speech markup languages into a single standard served as the starting point for requirements for SSML. It's clear that SSML tries to learn and leverage from the extensive work done on speech synthesis markup languages in the past.

Unlike SABLE and JSML, SSML markup is specifically designed to support both complete standalone SSML-only documents, and the embedding of SSML within other markup languages such as HTML and VoiceXML. SSML is particularly well suited for use with a VoiceXML wrapper when building an interactive voice response application. Additionally, SSML is designed to be easily embedded in multimedia SMIL presentations. SSML is also considered a superset of the Aural Cascading Style Sheets capabilities, with the notable omission of support for spatial audio.

Most features of SSML are designed to be accessible to non-speech experts, however, some tags, such as those for speech contour design (eg. `phoneme` and `prosody`) are intended to allow experts to apply and use their specialized knowledge.

The key design criteria for SSML were:

- *Consistency*: SSML must provide predictable control of voice output across different platforms and across different vendor's speech synthesis engines.
- *Interoperability*: SSML must be integrate easily with other W3C specifications including (but not limited to) VoiceXML, aural Cascading Style Sheets and SMIL.
- *Generality*: SSML must support speech output for a wide range of applications with highly varied speech content.
- *Internationalization*: SSML must support speech output in a large number of languages within or across documents.
- *Generation and Readability*: SSML must support both automatic generation and hand authoring of documents. The documents should be human-readable.
- *Implementable*: The SSML specification should be implementable with existing, generally available technology, and the number of optional features should be minimal.

SSML supports both coarse and fine-grain control of speech synthesis. The SSML vocabulary allows word-level, phoneme-level, and even waveform-level control of the output to satisfy a wide spectrum of application scenarios and authoring requirements.

The full set of SSML tags is shown below in Table 4.

Table 4: SSML Markup Tags

Element Function	Element Name	Element Type	Element Description
Document Structure, Text Processing and Pronunciation	speech	Container	The root element, which must define the language of the root document.
	lexicon	Container	Specifies (possibly more than one) external lexicon pronunciation document, which contains pronunciation information for tokens that appear in spoken text.

	meta	Container	Associates a string to a declared meta property (attribute-value pair). Such pairs could include information about caching an SSML document in a web context.
	metadata	Container	A container in which any information about the document can be placed, using any metadata scheme (although the W3C RDF standard is recommended).
	p	Container	A text structure marker representing a paragraph.
	s	Container	A text structure marker representing a sentence.
	sayas	Container	Define the type of text contained, and specify the level of detail in how it should be rendered. SSML does not enumerate the set of text types. Example types include date, time, phone, currency etc.
	phoneme	Container	Provides a phonetic/phonemic pronunciation for the contained text. Can optionally define an alphabet (eg. IPA) used to specify pronunciation.
	sub	Container	Define aliases which replace contained text for pronunciation (eg. World-Wide-Web Consortium for W3C)
Prosody and Style	voice	Container	Request a change in speaker voice. Can specify gender, age, language, name etc. to constrain choice of voice.
	emphasis	Container	Specifies that the contained text should be spoken with emphasis (aka prominence or stress). Optionally, the level of emphasis can be specified.
	break	Empty	Controls pausing or other prosodic breaking between words. Strength and length of break can be specified.
	prosody	Container	Controls pitch, speaking rate, and volume of speech output. Actual pitch contours for contained text can also be directly specified.
Other	audio	Container	Supports the insertion of recorded audio files in various formats, and the insertion of other audio formats in conjunction with synthesized speech output.

	mark	Empty	Places a marker into the text/tag sequence which allows applications to be notified of events in the processing and rendering of SSML documents.
	desc	Container	Can only occur within the content of an audio tag. Contains a textual description of non-speech audio. Where only speech output is supported by a synthesis engine, the desc is spoken in place of the actual audio.

Another significant advance over previous speech markup languages is the use of an XML-Schema (rather than the less constraining Document Type Definition (DTD) representation) to define the syntax of SSML. This allows the specification to explicitly define both elements and content values permitted within SSML.

Apple Speech Synthesis Manager

The Speech Synthesis Manager [1] (formerly known as the Apple Speech Manager) is the part of the Mac OS that provides a standard method for Macintosh applications to generate synthesized speech.

This Speech Synthesis interface is designed with very different goals to the speech markup languages, in that it is squarely targeted at dedicated application developers, who must access the specific functions in the Speech Synthesis API to synthesize speech and control speech parameters.

While there are functions to monitor and change the rate and pitch of speech, the ability to specify even pronunciation of text is severely limited, in comparison with the functionality offered by the markup languages outlined so far. For example, the reference manual [1] for the Apple Speech Synthesis Manager suggests:

Converting textual data into phonetic data is particularly useful during application development, when you might wish to adjust phrases that your application generates to produce smoother speech. By first converting the target phrase into phonemes, you can see what the synthesizer will try to speak. Then you need correct only the parts that would not have been spoken the way you want.

Clearly this is a very inefficient way of actually controlling pronunciation of speech output – instead of proactively guiding the synthesizer with metadata; a developer must alter the actual input text until it is converted into appropriate phonemes. There is, thankfully, also the possibility to load specific pronunciation lexicons to guide this pronunciation process.

An application can use any number of speech channel. Each channel maintains an independent set of voice and synthesis attributes (such as rate, pitch etc.). Speech channels can also be allocated into one of several modes. These modes determine how the synthesis engine attempts to articulate the input data. The available modes are:

- `modeText` – Default mode of attempting to pronounce text according to available lexicon(s).
- `modePhonemes` – A phoneme-processing mode where input text is interpreted as a series of characters representing various phonemes and prosodic controls.
- `modeNormal` – A mode where digits are assembled into numbers (so that 12 is spoken as “twelve”).
- `modeLiteral` – A mode where each digit is spoken literally (so that 12 is spoken as “one, two”).

Microsoft Speech API (SAPI)

Microsoft’s Speech API version 5 [11] introduced an XML based TTS markup language. Unfortunately, it is a completely proprietary language, closely tied to SAPI. In many ways it is less feature rich than SABLE, JSML and related standards. The SAPI5 TTS tags are classified into 5 categories:

- Voice State Control
- Direct Item Insertion
- Voice Context Control
- Voice Selection
- Custom Pronunciation

Voice State Control

The Voice State Control tags control voice volume, rate, pitch and emphasis for words or segments of text. There is also a `spell` tag for forcing a voice to spell out a word or text segment.

Direct Item Insertion

Three tags are defined in the direct item insertion class. These tags allow applications to “insert items directly at some level”.

The `silence` tag inserts a defined period of silence.

The `pron` tag allows an author to insert a specific pronunciation, although the phoneme specification language does not make use of the IPA, preferring instead to use English specific approximations of specific phonemes.

Finally, a `bookmark` tag functions exactly as the `mark` tag in JSML and SABLE, allowing an application to be notified when a particular section of marked-up text is reached by the speech synthesis engine.

Voice Context Control Tags

There are two tags in the voice context control class, and both act to provide additional contextual information to guide synthesis of speech.

The `PartOfSp` tag provides part of speech information for enclosed word(s). This functionality is limited, however, in that it only supports SAPI defined parts of speech, which include nouns, verbs, modifiers, functions, interjections and others.

The `Context` tag provides information about pronunciation of specific word classes (*contexts*) such as dates, currency and times. Unlike the `sayas` classes defined in JSML or SABLE, the set of contexts is not defined as part of the SAPI specification and is completely voice dependent. The SAPI documentation states that any string can be used as a context, but that each specific SAPI compliant voices may support a completely different set of contexts.

Voice Selection Tags

These tags are used to (potentially) change the current voice.

The `Voice` tag specifies a voice based on its attributes. Possible selection attributes include age, gender, language, name, vendor, and vendorPreferred. The `Voice` tag can be empty, in which case it changes the voice for all subsequent text, or it can have content, in which case it only changes the voice for that content. If no voice is found which matches the required attributes, then no voice change will be made (and the current or default voice will continue to be used).

The `Lang` tag selects a voice based purely on its language attribute.

Custom Pronunciation

Although the custom pronunciation class is listed under the SAPI XML TTS tags section of the documentation, it is geared towards recognition and response, rather than synthesis of speech. The example given: [`<P>/disp/word/pron;/</P>`] is merely a shorthand for recognizing a *word* when pronounced as *pron*, and to display *disp* in response.

Microsoft Speech Application Software Development Kit (SASDK)

Microsoft seems to be phasing out SAPI as a standard, instead choosing to focus on its Speech Application SDK (SASDK) [12], which is a set of development tools supporting SALT (Speech Application Language Tags) for speech related technology. The SASDK is particularly focused on incorporating speech functionality into web applications.

The SASDK supports two types of speech output – TTS synthesis (using SSML – based on the W3C draft from April 2002) and Recorded prompts. The recorded prompts are clearly aimed at telephony style speech applications, where speech output is fairly limited in scope. The recorded prompts require human speech to be recorded and stored in a database, which is then used to provide concatenated speech output in preference to TTS synthesized speech. TTS is mostly used as the fallback option for situations where pre-recorded speech is not available.

Microsoft has also defined a proprietary Prompt Engine Markup Language (PEML) to control the production of speech by concatenation of recorded speech. PEML allows such control as choosing which databases to extract speech from, which specific prompt should be used (without searching for matches) and the ability to divide the input text into segments which are searched for separately in the recorded speech databases.

VoiceXML (VXML)

VoiceXML is designed for creating audio dialogs that feature synthesized speech, digitized audio, recognition of both speech input and telephony based Dual Tone Multi-Frequency (DTMF) key input, recording of speech input, telephony, and mixed initiative conversations. Its major goal is to bring the advantages of web-based development and content delivery to interactive voice response applications.

The example shown in Figure 3 illustrates the basic structure of a VoiceXML document. The top-level element is `vxml`, which is mainly a container for dialogues. There are two types of dialogues in VoiceXML: forms and menus. Forms are for presenting information to a user, and gathering input from a user. Menus offer choices for what to do next. A field is an input field, which a user must provide a value for, prior to proceeding to the next form element. VoiceXML also contains a self-contained execution environment for interpreting VoiceXML markup at runtime. This is used to define and control the dialogue flow (the execution path).

Figure 3 shows a form with a single field being used to prompt a user with a question, then processing the response with a specific grammar and passing the result to a server script.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
    http://www.w3.org/TR/voicexml20/vxml.xsd"
  version="2.0">
  <form>
  <field name="drink">
    <prompt>Would you like coffee, tea, milk, or nothing?</prompt>
    <grammar src="drink.grxml" type="application/srgs+xml"/>
  </field>
  <block>
    <submit next="http://www.drink.example.com/drink2.asp"/>
  </block>
  </form>
</vxml>
```

Figure 3: Example of VoiceXML Markup (taken from [10])

The definition of a VoiceXML-like specification began in 1995, with AT&T's Phone Markup Language (PML). By 1998, Motorola (VoxML), IBM (SpeechML), HP (TalkML), PipeBeach (VoiceHTML) and others were all working independently on similar languages for dialogue design. The VoiceXML Forum was formed to bring together these fragmented languages and concentrate efforts on defining a standard

dialogue design language that could be used to simplify the construction of conversational, speech based applications. In 2000, the VoiceXML Forum publicly released version 1.0 of the standard. This was submitted to the W3C, which then worked on refining and augmenting the original VoiceXML standard to create a W3C standard. Version 2.0 of VoiceXML [10], released in 2004, is the result of this effort.

The scope of VoiceXML includes:

- Output of synthesized speech (text-to-speech).
- Output of audio files.
- Recognition of spoken input.
- Recognition of DTMF input.
- Recording of spoken input.
- Control of dialog flow.
- Telephony features such as call transfer and disconnect.

This is clearly a much wider scope than that covered by the speech markup languages already presented.

The `prompt` tag is the component of VoiceXML that results in speech output. In VoiceXML 2.0, the content of a `prompt` element can be marked up using the W3C SSML.

There are some slight variations to the SSML syntax allowed with VoiceXML. Firstly, the `speak` tag (being a root level tag) is not valid for use within VoiceXML `prompt` tags. Also, additional properties are defined for the `audio` and `say-as` elements. For the `audio` tag, additional attributes defined for VoiceXML include:

- *fetchtimeout* – Timeout for retrieving an audio file.
- *fetchhint* – Defines when the interpreter context should retrieve content from the server (eg. prefetch, fetch when needed).
- *maxage* – Indicates that the document is willing to use audio content whose age is no greater than a certain value.
- *maxstale* – Indicates that the document is willing to use content that has exceeded its expiration time.
- *expr* – An ECMAScript expression which determines the source of the audio to be played.

VoiceXML also extends the `sayas` SSML tag by adding ‘interpret-as’ values, corresponding to built-in types. The set of built-in type definitions includes: boolean, date, digits, currency, number, phone, and time. SSML does not define any mandatory or standard set of types for `sayas`, so this is a logical extension.

Because VoiceXML is specifically designed for telephony applications, it does not consider the possibility of concurrent textual or graphical data display. As a result, there is little separation of content and presentation. To address some of these perceived

limitations several vendors have cooperated to propose an alternate, but related, SALT standard for speech (and particularly telephony) applications.

The VoiceXML Forum has attempted to address this issue with its XHTML+Voice [2] specification, which is specifically designed to embed VoiceXML content within an HTML context. The focus of this work is to bring spoken language interaction (input and output) to the web, to provide truly multimodal interfaces.

Speech Application Language Tags (SALT)

SALT [14], like VoiceXML, has a much wider scope than speech synthesis alone. In contrast to VoiceXML, however, SALT is intended to provide a minimal environment for defining speech applications. It is restricted to the definition of data and behaviour specific to the speech interface, such as listening for speech input and specifying grammars for interpreting that input. All other data, such as the definition of forms and other elements, is left to the markup language (such as HTML) in which the SALT markup is embedded. As a result, SALT is specifically designed to integrate easily and directly into the current paradigm of server side web applications and client side web browsers.

The main top-level elements of SALT are:

- **<prompt>** for speech synthesis configuration and prompt playing
- **<listen>** for speech recognizer configuration, recognition execution and post-processing, and recording
- **<dtmf>** for configuration and control of DTMF collection
- **<smex>** for general purpose communication with platform components.

The input elements `listen` and `dtmf` also contain grammars and binding controls:

- **<grammar>** for specifying input grammar resources
- **<bind>** for processing recognition results.
-

`listen` also contains the facility to record audio input:

- **<record>** for recording audio input

`smex` also contains the binding mechanism `bind` to process message.

In terms of speech synthesis, SALT also supports the use of the W3C SSML to specify how the platform's speech synthesis engine should render text in speech. In almost all cases, however, such speech synthesis only occurs in voice prompts, which are often generated from the concatenation of pre-recorded speech. As a result, in many SALT applications, TTS markup is often not used.

References

1. Apple Computer, (2003), "Speech Synthesis Manager Reference", published February 1st 2003.
2. Axelsson, J., Cross, C., Ferrans, J., McCobb, G., Raman, T., Wilson, L., (2004), "XHTML+Voice Profile 1.2", VoiceXML Forum Specification, published 16 March 2004 at <http://www.voicexml.org/specs/multimodal/x+v/12/spec.html>
3. Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., Yergeau, F. (2004), "Extensible Markup Language (XML) 1.0 (Third Edition)", W3C Recommendation 04 February 2004, published at <http://www.w3.org/TR/REC-xml/>
4. Burnett, D., Walker, M., Hunt, A., (2004), "Speech Synthesis Markup Language (SSML) Version 1.0", W3C Recommendation, 7 September 2004, Published at <http://www.w3.org/TR/speech-synthesis/>
5. Hartman, R. (2004), "SALT: The Speech Application Markup Language", In Dr Dobb's Journal Special Windows/.NET Supplement, June 2004, pp. S1-S5.
6. Hirschberg, J., (1995), "Controlling Intonational Variation Using Escape Sequences in the Bell Laboratories Text-to-Speech System", June 22 1995.
7. Hunt, A. (2000), "JSpeech Markup Language", W3C Note, published 05 June 2000 at <http://www.w3.org/TR/jsml/>
8. Isard, A. (1995), "SSML: A Markup Language for Speech Synthesis", PhD Thesis, University of Edinburgh.
9. Kohler K.J. (1997) "Parametric control of prosodic variables by symbolic input in TTS synthesis", In van Santen J.P.H., Sproat R.W., Olive J.P., and Hirschberg J., (eds.) *Progress in Speech Synthesis*, Springer, New York.
10. McGlashan, S., Burnett, D., Carter, J., Danielsen, P., Ferrans, J., Hunt, A., Lucas, B., Porter, B., Rehnor, K., Tryphonas, S., (2004), "Voice Extensible Markup Language (VoiceXML) Version 2.0", W3C Specification, published 16 March 2004 at <http://www.w3.org/TR/voicexml20/>
11. Microsoft Corporation (2003), "Microsoft Speech SDK 5.1 Documentation", Edition dated March 3rd 2003, Downloaded in October 2004 from <http://www.microsoft.com/speech/download/old/sapi5.asp>
12. Microsoft Corporation (2004), "Microsoft Speech Application SDK", Version 1.0, Published at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/SASDK_getstarted/html/SDK_Welcome.asp, 2004.
13. Raggett, D. (1995), "HyperText Markup Language Specification Version 3.0", W3C Specification, published 28 March 1995 at <http://www.w3.org/MarkUp/html3/html3.txt>
14. SALT Forum, (2002), "Speech Application Language Tags (SALT) 1.0 Specification", Dated 15 July 2002, published at <http://www.saltforum.org/saltforum/downloads/SALT1.0.pdf>
15. Sproat, R., Taylor, P., Tanenblatt, M., Isard, A. (1997), "A Markup Language for Text-to-Speech Synthesis", In Proceedings of Eurospeech, Rhodes, Greece, 1997.
16. Sproat, R., Hunt, A., Ostendorf, M., Taylor, P., Black, A., Lenzo, K., Edington, M., (1998), "SABLE: A standard for TTS markup", ICSLP98,

17. Slott, J. (1996), "A General Platform and Markup Language for Text to Speech Synthesis", MIT Masters Thesis.
18. Sun Microsystems (1999), "Java Speech API Markup Language Specification, Version 0.6", October 12 1999, Published at <http://java.sun.com/products/java-media/speech/forDevelopers/JSML/>
19. Taylor, P., Isard, A. (1997), "SSML: A speech synthesis markup language", Speech Communication, no. 21, pp. 123-133, 1997.