SABLE: A STANDARD FOR TTS MARKUP

R. Sproat[1], *A. Hunt*[2], *M. Ostendorf*[3], *P. Taylor*[4], *A. Black*[4], *K. Lenzo*[5], *M. Edgington*[6]

[1]Bell Labs – Lucent Technologies, [2]Sun Microsystems, [3]Boston University,
[4]CSTR – University of Edingburgh, [5]Carnegie-Mellon University, [6]BT Labs

ABSTRACT

Currently, speech synthesizers are controlled by a multitude of proprietary tag sets. These tag sets vary substantially across synthesizers and are an inhibitor to the adoption of speech synthesis technology by developers. SABLE is an XML/SGML-based markup scheme for text-to-speech synthesis, developed to address the need for a common TTS control paradigm. This paper presents an overview of the SABLE specification, and provides links to sites where further information on SABLE can be accessed.

1. INTRODUCTION

There is an ever increasing demand for speech synthesis (TTS) technology in various applications including e-mail reading, information access over the web, tutorial and language-teaching applications, and aids for the handicapped. Invariably, an application that was developed with a particular TTS system A cannot be ported, without a fair amount of additional work, to a new TTS system B, for the simple reason that the tag set used to control system A is completely different from those used to control system B. The large variety of tagsets used by TTS systems are thus a problem for the expanded use of this technology since developers are often unwilling to expend effort porting their applications to a new TTS system, even if the new system in question is of demonstrably higher quality than the one they are currently using.[1]

SABLE is an XML/SGML-based [2, 1] markup scheme for text-to-speech synthesis, developed to address the need for a common TTS control paradigm. SABLE is based in part on two previous proposals by a subset of the present authors: the Spoken Text Markup Language (STML – [5]) and the Java Speech Markup Language (JSML — [4]).

The SABLE markup language is being developed with the following goals in mind:

- Synthesizer control: enables markup of TTS text input, for improving the quality and appropriateness of speech output.
- Multilinguality: the tagset should be appropriate for any language.
- Ease of use: SABLE should not require specialized knowledge of TTS or linguistics, though users with such experience should be able to apply their knowledge.
- Portability: provides application developers with a consistent mechanism for controlling synthesizers from different companies and on different platforms.
- Extensibility: SABLE should be able to evolve to support new features in future releases. Furthermore, to encourage research, SABLE should allow individual synthesizers to support enhanced features without compromising the portability of SABLE text.

SABLE, like its predecessors, supports two kinds of markup: the first – termed *text description* in STML, and *structural elements* in JSML – marks properties of the text structure that are relevant for rendering a document in speech. In the current version of SABLE, *text description* are handled by the DIV tag, whose attribute TYPE may be set to such values as *sentence*, *paragraph* or even *stanza*; and by SAYAS, which marks the function of the contained region (e.g. as a date, an e-mail address, a mathematical expression, etc.), and thereby gives hints on how to pronounce the contained region. The second kind of markup – STML's *speaker directives* or JSML's *production elements* – control various aspects of how the speech is to be produced. Falling into this latter category are tags such as: EMPH (marks levels of emphasis); PITCH (sets intonational properties); RATE (sets speech rate); and PRON, which provides pronunciations as phonemic strings.

In both its generality and its coverage, SABLE is superior to existing markups such as Microsoft's SAPI [3], or Apple's Speech Manager control set. Whereas the syntax of other schemes is typically ad hoc, SABLE's is based on XML/SGML, a widely-used standard. Secondly SAPI and other markup schemes provide tags only for speaker directives, not for text description. Text-description information, for example, that a particular boundary in a text corresponds to the end of a line in a table (e.g., <DIV TYPE=x-tl>), can in principle be used by a TTS system to advantage to produce reasonable speech output that marks auditorily the presence of that boundary. One does not necessarily want to have to instruct the synthesizer to use a particular intonation pattern, or to implement the break in a particular fashion: one might prefer simply to mark the presence of the boundary in a fairly abstract way, and assume that the system will do something reasonable with that information. Text-description is explicitly designed to allow that kind of abstract specification.

2. TAGS AND ATTRIBUTES

The draft specification of SABLE V0.2 contains the following tags and attributes; in many cases the meaning of the attribute is fairly obvious and we dispense with a description in such cases. Full descriptions of these, and other details, can be found at the URLs listed in Section 4.. Note that the terms *container element* and *empty element* are standard technical terms in SGML/XML:

[1]One might imagine that the industry "standard", Microsoft's SAPI, has solved this portability problem, but this is in fact not the case: only minimal requirements of compliance are required for a system to be able to claim that it is SAPI compliant, and thus there is no guarantee that the same set of tag specifications will yield comparable results when used with two distinct "SAPI-compliant" systems.

they denote, respectively, tags that consist of both a beginning and end tag, and thus *contain* enclosed text; versus tags that consist only of an end tag, and which are thus *empty* since they contain no enclosed text.

In addition to the attributes listed, every tag allows the MARK attribute, which sets an arbitrary mark. This can be used by specific SABLE-compliant TTS engines to report back to the calling application that it has reached the given location.

2.1. Speaker Directives

- EMPH (container element): set the emphasis of the contained text.
  - LEVEL (numeric, descriptive)
- BREAK (empty element): sets an intrasentential, prosodic break at current position.
  - LEVEL (numeric, descriptive)
  - MSEC (numeric)
  - TYPE (descriptive): a punctuation symbol that represents (roughly) the kind of intonation contour to be associated with the material preceding the break (e.g. '?' to mark "question" intonation).
- PITCH (container element): sets properties associated with pitch of the enclosed region.
  - BASE (numeric, descriptive)
  - MIDDLE (numeric, descriptive)
  - RANGE (numeric, descriptive)
- RATE (container element): sets the average speech rate of the enclosed region.
  - SPEED (numeric, descriptive)
- VOLUME (container element): sets the amplitude of the enclosed region in terms of the available range of the engine.
  - LEVEL (numeric, descriptive)
- AUDIO (empty): load and play an audio URL starting at the given point.
  - SRC: URL of audio document
  - MODE: specifies whether to play in background or not
  - LEVEL: level of audio document relative to synthetic speech
- ENGINE (container): substitute the DATA for the contained text if the system happens to be using the engine specified by ID.
  - ID: id for the TTS engine
  - DATA: character string to be substituted for the contained text
- MARKER (empty): anchor point for MARK attribute (see below) not otherwise associated with a tag.
- SABLE (container): specifies the document as a SABLE document.
- PRON (container): substitute the specified pronunciation for what would normally correspond to the contained text.
  - IPA: character string in Unicode IPA
  - SUB: attempt at "phonetic" spelling in the language of the enclosing text
  - ORIGIN: iso639 identifier for the language of origin of the enclosed text
- LANGUAGE (container): specifies the language of the contained text.
  - ID: iso639 identifier for the language
- SPEAKER (container): defines properties of the speaker speaking the contained text
  - GENDER
  - AGE (descriptive)
  - NAME: "name" of a speaker if a particular engine is being used

As an example of the use of some of these tags, consider the following example from a hypothetical e-mail reader that uses SABLE markup. Since e-mail readers have access to information about at least some structural aspects of the input — e.g. header information about the sender, subject and date, this information can be used to control the synthesizer's behavior in useful ways. For example consider the following marked-up example derived from the e-mail's header:

```
<DIV TYPE="paragraph">New e-mail from
<EMPH>Tom Jones</EMPH>
regarding <PITCH BASE=high RANGE=large>
<RATE SPEED="-20%">latest album</RATE>
</PITCH>.</DIV>
<AUDIO SRC="beep.aiff">
```

The subject information ("latest album") is highlighted auditorily by setting a higher base pitch and larger pitch range, and by slowing down the speech by 20%. Finally, the header is terminated by an audible beep (*"beep.aiff"*).

2.2. Text Description

- SAYAS (container): define mode in which contained text is to be said.
  - MODE: description of mode (e.g. date, time, phone, currency ...) in which contained text is to be read
  - MODETYPE: secondary specification further qualifying MODE (e.g., *date* is to read in order *day-month-year*)
- DIV (container): classifies the contained region as a structural text type of type TYPE
  - TYPE: type of the division (e.g. sentence, paragraph ...)

As an instance of the SAYAS tag, consider the rendering of the date “4/5/98” via the U.S. versus non-U.S. methods. These two ways of expanding the string can be specified as:

```
<SAYAS MODE=date MODETYPE=MDY>4/5/98</SAYAS>
```

and

```
<SAYAS MODE=date MODETYPE=DMY>4/5/98</SAYAS>
```

respectively. Specifications of this kind are unavailable in other inline markup schemes (e.g. SAPI), but they are useful to have since they have the potential to reduce cross-synthesizer inconsistencies. That is, without SAYAS specifications of the kind specified above, one cannot control whether a new system will pronounce “4/5/98” as “April 5, 1998” or “May 4, 1998”. The behavior for SABLE-compliant engines is, however, guaranteed.

2.3. Non-Standard Extensions to SABLE

SABLE is designed to function as a well-defined standard in which the same text will be handled consistently by multiple synthesizers. SABLE is also intended to function as a tool for research on speech synthesis and as a tool for innovation. As such, it is expected that research systems will support tags, attributes and attribute values not defined in the SABLE specification, and that SABLE text will be generated for specific systems which includes those tags and attributes. Where such extensions prove useful and become generally supported they can be proposed as an addition to the standard specification.

To clearly distinguish tags, attributes and attribute values that are non-standard they should include an “X-” prefix and optionally an engine identifier. A non-standard *tag* for providing an engine-specific pronunciation string would look like:

```
<X-ME-PRON PHON="i" DUR="120"/>
```

where ME is “My Engine” and the X-ME-PRON element inserts an /i/ phoneme with a duration of 120 msec understood by “My Engine”. Here, because the PHON and DUR attributes are embedded in a non-standard element, they are implicitly non-standard attributes. A non-standard attribute of a standard tag would look as follows:

```
<PRON X-ME-PHONES="ka:t">cat</X-ME-PRON>
```

or

```
<EMPH LEVEL="strong"
X-PITCHACCENT="H*+L">word</EMPH>
```

The first example provides the pronunciation for *cat* in a format that is understood by “My Engine”. Other synthesizers will ignore the attribute.

The second example includes both a standard attribute — LEVEL — and a non-standard attribute — X-PITCHACCENT. A system that understands the non-standard attribute will apply the “H*+L” accent when producing string emphasis on ”word”.

Finally, a non-standard attribute value might look like:

```
<DIV TYPE="x-dialog-close">...</DIV>
```

The “x-dialog-close” is a non-standard value of the standard TYPE attribute which is currently specified as being either “sentence” or “paragraph”. This non-standard value could indicate that the contents of the element are the end of a dialog turn.

If an engine gets a non-standard tag, attribute or attribute value in its input text that it does not know, it simply ignores it. For example, in the X-ME-PHONES example, a synthesizer that ignores the tag will try to say the word *cat*. Wherever possible, non-standard tags and elements should be designed so that output is not substantially impacted if ignored.

3. FURTHER ISSUES

This section addresses two issues: adding support for new TTS engines in SABLE, and generating SABLE markup for multiple TTS engines.

3.1. Adding support for a new TTS engine

SABLE may be implemented as either an SGML-based language or as an XML-based language. XML is a subset of SGML offering a simpler definition of markup languages that are easier to implement.

As most synthesizers already have their own idiosyncratic markup language, the simplest method for implementing a SABLE interpreter is by translating the SABLE tags directly into the particular synthesizer’s own markup escape sequences. There a number of free XML parsers and such translations can even be done directly in languages such as PERL. A complete translation of the file in a pre-processing stage will be sufficient for many cases, though when the input files are very big, a more integrated approach may be required.

When a particular engine does not support a tag it can (mostly) ignore the tag and simply synthesize the text contained within it. The tags are designed such that this is a reasonable fallback position as it is well known that different TTS engines may support quite different functionality. Where ignoring tags becomes problematic is in the SPEAKER and LANGUAGE tags.

We have allowed an engine-independant mechanism for specifying speakers but when a desired GENDER/AGE does not exist, the implementation should make a reasonable decision. Often a document may simply require one voice, plus an alternate voice the gender/age of which is not crucial. Although the NAME attribute may be used to specify a speaker by name, that is usually not going to work across engines: one possible solution — one which has not yet been agreed upon — is to allow a few standard names that an implementation should normally define,

e.g. MALE1, MALE2, FEMALE1, FEMALE2, and VOICE1, VOICE2 when gender is not relevant. Such a mechanism is more likely to work across engines.

Dealing with the LANGUAGE tag when an engine does not support that language is a more difficult task. In the *Festival* implementation of SABLE, for example, the system simply says "Something in X" when a section of text is within marked as being of language X, where *Festival* does not support that language; there is of course some question about which language this comment should be said in.

3.2. Generating SABLE markup for multiple TTS engines

By using the ENGINE tag it is possible to make use of engine-specific functionality but our perceived use of SABLE is where the markup text is generated quite independantly of the engine used to render it as speech. In fact, one can imagine the synthesis engine running on a local machine while the text is generated by some web-based application elsewhere on the net: the advantage of this is that relatively low bandwith will be required between the machines and high quality audio can be generated locally.

When writing applications that generate SABLE markup it is fairly easy to write in a TTS-engine-independant way. Using relative and descriptive values for attributes such as the PITCH and RATE tags is much more likely to work across synthesizers than absolute values. Also given that engines may potentially ignore tags when they do not support the functionality, the text can be arranged such that this will still sound reasonable.

As we primarily see SABLE markup being generated by applications rather than written by hand, using an existing standardized markup paradigm makes this much easier. We also envisage stand-alone applications which can translate existing documents into SABLE markup. For example, e-mail messages are structured, and a convertion program could be written that marks headers, quoted sections etc. using SABLE tags that would allow any SABLE-compliant TTS engine to render it reasonably. Converters for existing document formats such as LaTeXand MS Word could also be written.

In the case where the document format is already an XML/SGML type language, such as HTML, existing document translation systems can be exploited. HTML may be augmented with cascading style sheets (CSS) (**http://www.w3c.org/css/**) which would make translation to SABLE very simple.

4. FURTHER INFORMATION

A number of SABLE-related resources are publicly available. The latest draft of SABLE, along with the latest SABLE XML DTD, can always be found at the following web addresses:

- **http://www.bell-labs.com/project/tts/sable.html**
- **http://www.cstr.ac.uk/sable.html**

One can also find on-line demos of subsets of SABLE at those sites.

Experimental tools for SABLE are also available at the Edinburgh site: XML-based parsers/interpreters are available to interface between SABLE and *Festival*, and between SABLE and the Bell Labs/Lucent Technologies TTS system. The tools can be downloaded and adapted for one's own favorite TTS system, or else they can be used as the basis for developing one's own SABLE application.

Finally, there is an e-mail discussion group for SABLE. To join this, send a message to **sable-subscribe@east.sun.com**. To succeed as a standard that benefits both commercial and academic users, it is important for SABLE to be designed with input from many sources. In addition to informing the speech community of the status of the SABLE initiative, it is our hope that this paper will increase participation and interest in its development.

5. REFERENCES

1. Consortium, W. W. W. Working draft: Extensible markup language (xml)version 1.0 part 1: Syntax. **http://www.w3.org/TR/REC-xml**, 1998.
2. Goldfarb, C. *The SGML Handbook*. Oxford, Clarendon Press, 1990.
3. Microsoft. *Microsoft Speech Software Development Kit Developer's Guide*, version 2.0 ed. Microsoft, Redmond, WA, 1996. Version 2.0.
4. Microsystems, S. Java Speech Markup Language specification. **http://java.sun.com/products/java-media/speech/**, 1997.
5. Sproat, R., Taylor, P., Tanenblatt, M., and Isard, A. A markup language for text-to-speech synthesis. In *Proceedings of the Fifth European Conference on Speech Communication and Technology* (Rhodes, 1997), ESCA.