Implementing an SSML Compliant Concatenative TTS System

Andrew Breen, Steve Minnis and Barry Eggleton

Nuance Communications Inc.
Menlo Park, CA

andrew.breen@nuance.com, steve.minnis@nuance.com, barry.eggleton@nuance.com

Abstract
The W3C Speech Synthesis Markup Language (SSML) unifies a number of recent related markup languages that have emerged to fill the perceived need for increased, and standardized, user control over Text to Speech (TTS) engines. One of the main drivers for markup has been the increasing use of TTS engines as embedded components of specific applications – which means they are in a position to take advantage of additional knowledge about the text. Although SSML allows improved control over the text normalization process, most of the attention has focused on the level of prosody markup, especially since the prediction of the prosody is generally acknowledged as one of the most significant problems in TTS synthesis. Prosody control is by no means simple due to the large cross-dependency between other related aspects of prosody. Prosody control is also of particular complexity for concatenative TTS systems. SSML is about much more than prosody control though – allowing high level engine control such as language switching and voice switching, and low level control such as phonetic input for words. Our experiences in implementing these diverse requirements of the SSML standard are discussed.

1. Introduction
The W3C SSML 1.0 specification [1] has received much attention recently [2] yet few if any TTS engines offer full SSML compliance. This is due to various reasons. One is that an SSML implementation requires not only the (embedded) TTS engine compliance, but also the TTS controlling application compliance. For example, SSML allows TTS voice switching, TTS language switching, and the interleaving of audio with the TTS output. In addition, our aim is not simply to be compliant but to produce effective results, this is particularly difficult with respect to the prosodic flags. The SSML standard for example, allows a user to specify the location of a prosodic break via the <break> flag, as shown in following examples taken from the standard.

3. Nuance Vocalizer 3.0 Architecture
In order to describe our implementation, Figure 1. shows a simple schematic diagram of our TTS system, Nuance Vocalizer 3.0.

A controller manages a series of TTS engines (only one is shown). The controller also manages the flow of input text (2) to engines, piping it through appropriate preprocessors where required (not all text will be marked up in SSML). It is possible to chain preprocessors; so an Email may contain

However, the standard does not specify how this flag interacts with the breaks that would be predicted by the engine on the same text. Normally a break would have associated prosodic realizations, for example increased duration of the syllable before the break, and heightened pitch changes. How would these effects interact with the use of a combination of manually inserted break, duration and pitch contour controls? Clearly the standard is geared mainly at specifying normalized input, and does not have so much to say about how this input is to be interpreted apart from the say-as flag which attempts to address this.

This paper discusses our implementation and outlines our interpretation of the input, particularly with respect to the prosodic markup flag.

2. The SSML Standard
The W3C SSML 1.0 specification [1] is intended to bring together in one standard related markup languages such as Sun Microsystems JSML [3], (Edinburgh) SSML [5] and SABLE [5].

The SSML standard has come under close scrutiny from the TTS community [6]; a common complaint is that the standard mixes high level user control and specialist low level control. It has also been highlighted that the standard does not provide guidance on how the levels of control should be applied, for example both the emphasis and prosody/contour flags could be used to signal prominence. What is the best way? How should they best be combined, if at all. Should/Could the standard address these issues? At Nuance we sought to undertake a full implementation of the standard to explore these issues.

There are two main sections to the standard related to (1) Document Structure, Text Processing and Pronunciation, and (2) Prosody and Style. A third section deals with higher level control, such as embedding audio files, and providing the calling application with callback markers into particular points in the synthesized speech.

The three-way classification of the SSML standard initially seems like a reasonable distinction to make, but in reality it mixes control at a variety of levels. These will be illustrated by describing our architecture.

Take a deep breath &lt;break/&gt; then continue.
Press 1 or wait for the tone. &lt;break time="3s"/&gt; I didn't hear you!

However, the standard does not specify how this flag interacts with the breaks that would be predicted by the engine on the same text. Normally a break would have associated prosodic realizations, for example increased duration of the syllable before the break, and heightened pitch changes. How would these effects interact with the use of a combination of manually inserted break, duration and pitch contour controls? Clearly the standard is geared mainly at specifying normalized input, and does not have so much to say about
The marking of date class, allows the parser to treat this as a special type of noun type. This is necessary since linguistic recovery of information is difficult to do reliably.

The <phoneme> flag is not as straightforward to implement as it might seem, due to the need for intra-alphabet translations. Should phonemes in one alphabet that have no direct corresponding mapping in another map to silence or to the nearest appropriate phoneme? What determines this appropriateness?

Also at the front end we need to consider the <break> and <emphasis> elements. Interestingly enough these are divorced from the <prosody> flag in the standard, although both are directly related. They fall under our front end/back end classification neatly, since they are both directly concerned with the symbolic representations in our TTS engine.

The <break> flag is, according to the standard “most often used to override the typical automatic behavior of a speech synthesizer” ([1], section 2.8). This scenario implies a post-processing role – a way of changing the pausing and prosodic structure of the sentence. This is how we have implemented the flag in Nuance Vocalizer 3.0.

A post-processing implementation avoids the problems associated with the incorrect implicit assumption that break insertion has only a local effect. Prosodic break segmentation is usually performed using algorithms where many factors interact (see [10] for a summary of such algorithms). For example Wang and Hirschbergs' decision tree implementation had syntactic structure, as well as time and number of words since the last break [7]. Similarly in Taylor and Blacks' n-gram model break positions are interdependent [8]. Implementations of rule based approaches also have the same interactions [9]. Break insertions are not local in practice.

Break insertion also has an effect on the perceived emphasis in the sentence, since (generally) nuclear stress would occur on the last content word before a break. Our method of implementing the <emphasis> flag (with value="large") is to shift the nuclear emphasis onto another word in a prosodic phrase (simply defined as a group of words delimited by two large pauses). Needless to say the potential for nonsensical markups (emphasizing all the words in a sentence) requires fallback to some sensible realization, and in some cases the expected behavior will be constrained by the internal system parameters.

The <break> and to some extent the <emphasis> elements also interact with the prediction of other prosodic elements such as the boundary tones and pitch movements across the sentence. However, all further control of pitch and contours is via the SSML <prosody> elements, which interact with the TTS back end components.
5. TTS Backend (Prosody) Flags

The TTS backend component takes as input the symbolic representation of all the required elements to synthesize the input sentence. Word, syllable and phoneme information is known, as well as relative salience, stressing and prosodic structure. In a concatenative synthesis system, the backend component typically includes Unit Selection and Realization components. Based on the symbolic input, the Unit Selection component selects sequences of naturally contiguous phonemes (collectively known as units) from an extensive and appropriately labeled speech database, through the optimization of a number of cost functions. Whereupon, the Realization component applies time/pitch scaling to the units and concatenates them to form a seamless completed utterance.

5.1. Performance, Naturalness and Quality

The backend of a concatenative TTS system entails a finely tuned tradeoff between performance, naturalness and quality. Natural, i.e. human-like, speech is achieved through: 1. the selection of units that match the phonological context of the target utterance as closely as possible; and 2. judicious use of ‘corrective’ prosodic (duration and pitch) modification within the Realization component. Any amount of signal processing applied to the units to correct or otherwise modify their intrinsic duration or pitch will lead to a reduction in speech quality. Furthermore, signal processing is also a cpu-intensive task, and therefore should only be applied where necessary.

The SSML flags available to a user to manipulate low-level prosody are perfectly capable of adversely affecting both the naturalness and quality of the rendered speech, not to mention the performance of the TTS engine. There now follows a discussion of the issues in interpreting and implementing some of the more problematic SSML prosody flags.

5.2. Duration

The duration flag is used to specify the total duration (ms) of the rendered speech contained within the flag range. A user cannot accurately know the intrinsic duration of the speech. Therefore, the amount of time stretching or compression that will be perceived cannot be known in advance - unlike the rate flag where the amount of duration modification is explicitly stated, being the inverse of the rate parameter. It is our interpretation that the duration flag is to be used in a wholly different manner to the rate flag, which is typically used to slow down the rendering of important information such as digit sequences. We infer that it is to be used for precise synchronization with other events occurring within a controlling system. This implies that a high level of accuracy is required from the flag, and herein lies the problem.

It is sensible to assume that a uniform time stretch/compression is required, applied to all phonemes within the flag's range so that the requested duration is realized. Therefore the total intrinsic duration of the complete rendered flag range must be known a priori so that an appropriate time stretch factor can be calculated and applied to the constituent units during rendering. It would be a simple matter to sum the units intrinsic durations, were it not for the fact that certain sounds are more readily modeled than others due to the intrinsic acoustic properties of particular sounds. In addition, the process of joining sounds inevitably leads to errors in the predicted total duration, and the degree to which any acoustic information applies to the joining process varies with the phonetic properties of the units' phonemes.

In short, each unit join is a special case, and each join requires a different amount of unit signal processing. Unless compensative measures are taken, each unit join affects the total duration of the rendered utterance, and the longer the utterance the greater the reduction in accuracy. So to accurately calculate the intrinsic duration of the flagged range before the flag is applied is a complex process, at worst requiring the concatenation process to effectively be performed twice.

So the TTS developer when implementing this flag needs to decide on a tradeoff between accuracy and performance, but insufficient accuracy may render the flag useless for applications requiring tight synchronization.

5.3. Pitch

The pitch flag is used to specify absolute or relative changes to the fundamental frequency of the phonemes within the flag's range.

Relative pitch changes pose no significant implementation problems. Providing the intrinsic pitch of each frame in the utterance can be accurately extracted, the new pitch can be calculated and reapplied. Simple transposition of fundamental frequency preserves the intrinsic pitch movement of natural-sounding speech as long as the pitch extraction and re-application frame length is no greater than a few pitch periods. The TTS engine must impose limits on the requested deviation from the intrinsic pitch to avoid unnatural sounding speech, and to minimize signal-processing artifacts.

Problems arise, however, in the interpretation of absolute pitch change requests. What exactly does a user expect from:

<prosody> pitch="120Hz" This is a test sentence </prosody>?

Interpreted literally, the flag would set the fundamental frequency of every frame of voiced speech to 120Hz. This would result in an unnatural, robotic monotone. On the other hand, perhaps the flag should be interpreted such that the average pitch across the utterance should be 120Hz. In which case the pitch range may still be as variable as without the flag. We chose to interpret the flag’s meaning as being somewhere between the two by transposing the pitch on a per syllable basis, such that the average pitch of each syllable is set to the requested frequency. This way the intrinsic, natural pitch movement is preserved, yet the flag’s effects are obvious.

5.4. Contour

The contour flag allows a user to specify pitch movement across the flag's range using interval/value pairs, where interval is a percentage of time through the flag's range, and value is as per the pitch flag.

The value parameters suffer from the same problems as the pitch flag, but the interval parameters make the flag’s use
awkward at best, and practically unusable for any flag range longer than a single word. The problem for a user is knowing, during text preparation, how values of interval relate to the positions of phonemes and syllables within the flag’s range. This is extremely difficult to judge simply by examining the text, and typically requires excessive trial-and-error. If the user’s intent is to stress an intrinsically unstressed syllable, improper placement of the contour could create unnatural, undesirable or impossible intonation.

The contour flag also highlights the same implementation difficulties found with the duration flag. As the interval parameters are specified as percentages of time through the flagged utterance, the total duration of the flagged range needs to be accurately calculated before it can be rendered with the prosodic modifications incorporated.

The contour flag makes it notoriously difficult for an unskilled user to achieve natural sounding results. We believe that low-level ‘contour crafting’ should generally be avoided in favor of the higher-level controls such as the <emphasis> flag, question marks, exclamation marks and parenthesis, where the TTS engine itself can ensure consistent naturalness and quality.

6. Discussion

Conformance can be measured in two ways. The first is adherence to the XML DTD defining the standard. This is easily done. The second is conformance to the functional specifications of the elements. This is less readily proven. Clearly different TTS engines will be able to effect changes to certain elements more readily than others.

Our Nuance Vocalizer 3.0 implementation backs off in a graceful fashion – if voices, languages or phonemes are specified that the system clearly cannot handle the input is ignored. A lot of this checking can take place at the Vocalizer controller level. If parameters are specified that exceed reasonable boundary limits, the maximum values or minimum values can be inserted.

In a standard which leaves such a wide interpretation on the acceptable output, it is inevitable that implementations of the standard will vary in their actual output. We recommend that the SSML standard provide more guidance on implementation – despite the known problems regarding the interdependence of break/emphasis/prosody elements, it is possible to perceptually prove compliance that a break exists, or that a word is emphasized. General acoustic correlates for these scenarios could be defined. Whether it is worth doing so is another matter.

7. Conclusion

At Nuance we have implemented what we believe is full compliance with the SSML standard. We have made certain implementation assumptions:

- some element markup parameters will be constrained by practical limits;
- the break flag does not affect other break locations, it is complementary
- the use of multiple flags which potentially interact is handled in a principled manner. The break flags apply before emphasis flags, and finally the prosody flag.

Ultimately in such a customizable scenario, sensible user discrimination in the use of SSML flags must prevail in order to achieve sensible effects.

8. References