SSML: A Speech Synthesis Markup Language

Paul Taylor and Amy Isard
Centre for Speech Technology Research

Centre for Speech Technology Research University of Edinburgh,
80 South Bridge Edinburgh EH1 1HN
email
Paul.Taylor@ed.ac.uk

Abstract

This paper describes the *Speech Synthesis Markup Language*, SSML, which has been designed as a platform independent interface standard for speech synthesis systems. The paper discusses the need for standardisation in speech synthesizers and how this will help builders of systems make better use of synthesis. The SGML based markup language is then discussed, and details of the Edinburgh SSML interpreter are given as a guide on how to implement a SSML-based synthesizer.

1 Introduction

This paper presents work we have carried out in the area of speech synthesis standardisation. Currently, most speech synthesis systems act as stand alone devices which are incompatible with one another and do not share common interface standards. As the general usage of speech synthesis increases, lack of standardisation will become a serious problem hampering the ease of integration of speech synthesis with other types of system. This paper addresses a specific issue in speech synthesis standardisation, namely the provision of a standard text annotation scheme.

1.1 Annotated Text in Speech Synthesis

The most common type of input to a speech synthesizer is plain text. This is desirable in some sense as it is a commonly agreed format, understandable to everyone. However, text is not ideal because it is extremely difficult for a computer to automatically analyse the text and discern the discourse peculiarities of a sentence. In addition there are cases where the same sequence of words can be spoken in different ways, each being perfectly acceptable in a given context. Choosing the correct pronunciation can prove extremely difficult given the range of pragmatic factors which can influence the context of the sentence. If a synthesis system does not have access to such information, the speech typically sounds bland and often the wrong words are emphasised, leading to errors in intelligibility. To tackle this problem, many research and commercial systems allow for annotations in the text which allow direct control of aspects of the speech synthesizer's operation. Thus words can be emphasised, phrase breaks can be placed and instructions can be given which indicate how words should be pronounced.

The problem with annotation is that it loses the attractiveness of the text approach, namely standardisation. While text is familiar to everyone and doesn't vary from system to system, each synthesis system has its own annotation schemes. This means that users have to learn a different scheme with each synthesizer they use. More importantly, it is a headache for the system builder in that programs which interface with speech synthesizers must be altered if a different synthesizer is to be used. To tackle this problem, we have designed a speech synthesis markup language called SSML, which is aimed at becoming a standard speech synthesis interface.

We envisage several key areas in which SSML will help in making synthesis systems more usable. It is possible for SSML to be used in much the same way as the input to a normal text-to-speech system in that users can type the messages they wish to be spoken. However, we see the main use of SSML being in the synthesis of machine-generated messages. It is nearly always the case that systems which generate messages automatically have access to more information than just the basic words. This holds true for simple slot and filler style systems as well as more complicated linguistically sophisticated systems, such as those used in machine translation. Such systems can make use of their additional information by using tags to control pronunciation. The advantage that SSML brings is that the designers of such language generation systems need only understand the basic SSML language and do not need specialist speech synthesis knowledge. They are also free to use any SSML-compatible synthesis system they wish and are not tied to a particular synthesizer.

1.2 Position of Tags in Linguistic Description Space

There are many possible formats for the input to a speech synthesis system. Among the most common are plain text, phonetic descriptions (possibly enhanced with prosodic markings) and low level parameters such as formant tracks or linear prediction parameters. These range from inputs easy to understand by humans but difficult to interpret by computers, to more exact parameters, which may be unambiguous and easy to process by machine, but which are difficult for humans to produce.

Plain text is the most desirable form of input from a human perspective due to its standard nature and universal understanding. However, text is very difficult to process as automatic systems do not have access to the real-world information that helps in the disambiguation and processing of sentences. Hence, it is desirable to find other forms of input which make the text analysis task easier. Possibilities include: text with annotations, syntactic trees where possible

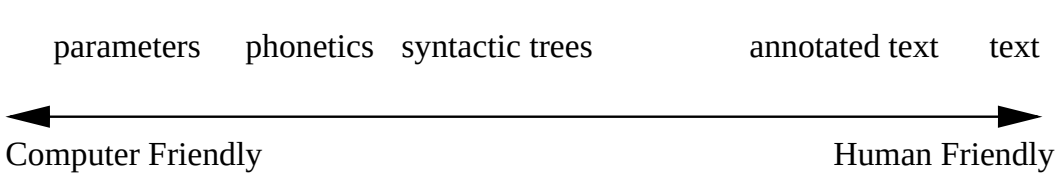

Figure 1: Impressionistic plot of different types of synthesis input

ambiguity in structure has been removed, semantic input with features such as tense being represented by variables, phonetic descriptions where the pronunciation of each sentence is exactly specified, phonological descriptions which are similar to phonetic ones but with less redundancy, and low-level synthesis parameters. These types of input can be analysed by comparing their "computer-friendliness" against their "human-friendliness". These terms are somewhat vague, but by computer-friendly we mean inputs which are unambiguous and require little real world knowledge for interpretation, and by human-friendly we mean inputs which can easily be written and (understood) by a person. An additional important point which must be made is that the perceived human-friendliness of a input may be heavily dependent on the individual: a linguist will find a syntax tree less daunting than a user with no linguistic expertise. It must also be noted that here the term human-friendliness is only an indication of the ease of writing the input to synthesis systems, and should be differentiated from human-satisfaction which is a measure of the overall end quality of the system. Figure 1 gives an impressionistic plot of the types of input.

The position of our standard in the space of possible inputs is thus a *design decision*. It is clear that a single standard cannot hope to please everyone. In essence, the decision involves choosing who the potential users will be and what degree of synthesis control is desired.

SSML's position is close to that of those TTS systems which allow annotations in their inputs. SSML is thus aimed at users who have some computer literacy, but who aren't experts in linguistics. Some investigation has been directed towards the development of an "expert" SSML for experienced users and/or linguists, but this has so far proved difficult, mainly due to problems with choosing a sophisticated linguistic description system that is acceptable across different linguistic theories.

1.2.1 Overview of the Paper

The layout of the paper is as follows: section 2 describes SGML, the language in which SSML is written. Section 3 describes SSML version 1.0 and discusses implementation issues of certain

SSML features. Section 4 describes the Edinburgh markup-to-speech (MTS) system. In this section we describe how we have chosen to implement our SSML interpreter. Builders of other synthesis systems are free to interpret SSML features as they feel fit, and need not construct their systems in the same way we have described here. We have included this section simply to give guidelines on how SSML interpretation may be carried out.

Section 5 describes some situations where we think the adoption of SSML as a standard would be particularly beneficial. We describe how SSML helps in the coupling of synthesizers to automatic language generation systems, and how filters can be employed to convert from visual-based markup languages into SSML. Section 6 discusses some newer, unimplemented features of SSML.

2 Using the Standard Generalised Markup Language

2.1 The Concept of Generalised Markup Languages

The *Generalised Markup Language* was first introduced by Goldfarb et al (1970). The intention was to produce a standard way to markup text in electronically stored documents so that they could be formatted and subsequently printed on a variety of machines and operating systems. From this initial work, a formal definition appeared called the *Standard Generalised Markup Language* (SGML) adopted as ISO 8879.

The machine/platform independence of SGML is based on the principle that the *logical* and *physical* aspects of a document's structure should be separated. The author of a document should specify the logical aspects, such as where headings occur, and the publisher (either a human publisher or automatic formatting program) should specify the physical aspects, such as which font to use for a heading. Although it is often not straightforward to follow this paradigm rigorously, the concept of logical and physical separation has proved extremely useful in providing machine independent authoring tools.

2.2 SGML Languages

SGML is not a markup language itself, but rather defines a way of creating markup languages. SGML defines a particular way of specifying markup syntax. The Speech Synthesis Markup

Language, SSML, described here, is an instance of a SGML language[1]. SGML is now widely used and is hence the most common basis for designing new markup languages. There is a wide selection of software tools available for authoring, validation, parsing and formatting of SGML documents.

As far as end users are concerned, the most noticeable feature of SGML is its syntax. Of course, personal preference will always play a part in opinions on the usability and aesthetics of a computer language and SGML has had certain criticisms, the main complaint being that its tags are too verbose. This point may be valid but we take the view that needing a few extra characters in the provision of tagging information is a small price to pay given the benefits of adopting the SGML paradigm. In any case, SGML-style syntax is becoming more familiar to average computer users. In recent years the SGML-based Hyper Text Markup Language, HTML, has been widely adopted, and thus SGML-like syntax will be familiar to a large number of users.

2.3 SGML documents

An SGML document consists of two sections:

1. A list and description of the elements which may be used and rules for how they can be combined

2. Some text which has been marked up using the defined elements

The element description is called the *Document Type Definition* (DTD), and the text section is called the *Document Instance*. When an SGML document is written, the DTD is normally put in a separate file, and a pointer to this file is included at the top of the document instance. This is partly for clarity, but mainly because in this way, the same DTD can be used for many different document instances.

The DTD also comes in two parts: the names and combination rules for the elements and a list of the information which can be contained within an element. The first of these is the *Element Declaration* and the second is the *Attribute Definition*.

[1] It should be made clear that most of what is described in this paper could be implemented in a different markup system to SGML; there is nothing in the fundamental concepts of SSML which necessarily requires it to be based on SGML.

Each element has a unique name, and a set of minimisation rules. The beginning of an element is signalled by a start-tag and the end by an end-tag. The minimisation rules determine whether or not start- and end-tags must be present in every occurrence of the element.

The attribute list for an element contains information which will be included in its start-tag. An element may have a number of attributes, each of which will have its value set every time the element is used in the document instance. There may be a range of values from which one is selected, or unconstrained text may be used to specify the value. It is also possible to set a default value.

3 Design Issues in Speech Synthesis Markup

3.1 Separation of Logical and Physical Aspects

Given that SSML has been chosen to be similar in scope to the annotation systems found in other TTS systems, the main problem is to define a set of annotations such that they are *platform-independent*. It is in trying to solve this problem that we adopt the same strategy used in the field of document processing, namely the separation of logical and physical aspects of the input.

Thus the problem of defining SSML comes down to specifying a set of tags which most systems will be able to understand and process, and doing so in a manner such that the resultant synthetic speech from different systems is judged to be roughly the same for a given input.

This specification has strong implications for the role of linguistic theory and descriptions within SSML, in that only widely accepted theories and description systems can be used. As an example, we will discuss the issue of intonation control within SSML. This example will also serve to demonstrate the level of expertise demanded of an SSML author.

3.1.1 Intonation specification in SSML

In English, a number of intonation models have been proposed (Pierrehumbert, 1980), (Crystal, 1969), (O'Connor and Arnold, 1973), (Ladd, 1983), (Taylor, 1992). At present, the tone based model of Pierrehumbert is the most popular and has been incorporated in an adapted form into the ToBI prosodic transcription system (Silverman et al., 1992). It might therefore seem that an intonation markup system should be based on such a model, but for reasons we will now go into, it was decided not to base the intonational control of SSML on a model such as Pierrehumbert's.

Although this system is the most widely used, its usage is not universal. This has two

implications. Firstly, many synthesis systems do not base their internal synthesis model on the Pierrehumbert system and therefore it may be very difficult for them to interpret tone-based markup tags. Secondly, the non-universal acceptance of the theory by linguists may be an indication that there are problems with the system which will result either in amendments or in a completely new system coming to light in the near future. Thus it may be short-sighted to lock SSML into a particular system such as this.

More importantly, we believe that even if the tone-based model were accepted more fully, it would probably still not be adopted as the means of intonational control within SSML. The tone-based system is not particularly accessible to the non-linguist and thus would be overly complex for the intended users. Although the intonational transcriptions of the British school of Crystal (1969) and O'Connor and Arnold (1973) are perhaps more accessible than tone-based models, even they are probably too complex.

The solution to the intonation problem came from work carried out in designing the input specification for the CHATR speech synthesis system (Black and Taylor, 1994b), (Black and Taylor, 1994a). This input consisted of marking words as having emphasis or not, and of also specifying a global tune type for each sentence. Thus words are tagged with a single type of marker which indicates they will receive a pitch accent, and it is the tune specification which decides which type of pitch accents these words will get.

We have found that this method allows the realisation of a wide range of intonational effects in a manner that is accessible to non-expert users. In addition, the class of tunes is easily extendible.

3.2 Set of Example Tags

Here we will explain the tags currently defined in SSML.

3.2.1 Phrasing:<phrase>

This tag can be inserted anywhere in the text to indicate the presence of an intonational phrase boundary. More recent developments in SSML have expanded the functionality of this tag to allow attributes to be associated with it. An attribute allowing control over the *degree* or *level* of phrasing has been added. This is a number which follows the break index system of the ToBI scheme. In contrast to the intonational component of ToBI, the break index component is relatively theory neutral, and in addition seems quite easy for naive users to pick up. If no

level is specified, the default is taken to be 4, which indicates an intonational phrase break.

A second type of attribute specifies the intonational tune of the phrase. At present allowable tunes include *statement*, *wh-question*, *yn-question* and *imperative*. If no tune is specified, the default is take to be *statement*.

3.2.2 Emphasis:<emph>

This tag is used to give emphasis to particular words. The last emphasised word in a phrase is taken to be the nuclear accent. The position of the emphasis indicates which particular word is to be emphasised, the speech-act specification from the current phrase indicates what the global tune of the phrase should be and together these can be used to construct an intonational specification for the phrase.

3.2.3 Pronunciation:<define>

Most TTS systems provide a pronunciation (i.e. phonemes and stress markings etc) for a sequence of words by a combination of lexical lookup, letter-to-sound rules and perhaps morphological analysis. These processes can fail to produce the required pronunciation of a word, and hence a facility has been added for specifying how a word should be pronounced at any point in the document. It should be noted that even very sophisticated automatic pronunciation systems will have difficulty in certain circumstances - see the example in the Appendix.

SSML defines word pronunciations by adopting a sort of one-off lexicon entry. SSML cannot assume a fixed definition structure for lexical entries as different TTS systems have different ways in which they specify the structure of items in their lexicons. To combat this problem, SSML allows users to specify pronunciations in any lexicon style they wish, so long as the name of the lexicon is given. The idea is that the synthesizer can convert from an explicitly named external format to its own internal format.

The accuracy of this conversion is dependent on the individual ability of the synthesizer to map from one format to another. Many lexicon formats differ in only a trivial manner and in this situation, no information should be lost. However, in cases where the two formats differ more widely, loss of information may be unavoidable.

In SSML, definitions are written in the following form:

<define word=*headword* pro=*pronunciation* format=*lexicon format*>

headword matches the word in the text that is to have its pronunciation defined. *pronunci-*

ation is a string which matches the format in the lexicon. *lexicon format* is optional and simply names the lexicon standard. A default, global *lexicon format* can be defined which will be used in the absence of this in the specific define command. For example, in the CSTR lexicon format the following would occur:

<define word= "edinburgh" pro= "e 1 d i n b 2 @ r @@" format= "cstr">

While the definition from the Carnegie Mellon Pronouncing Dictionary (cmudict.0.1) would be as follows.

< define word= "edinburgh" pro= "EH1 D AH0 N B ER2 OW0" format= "cmudict.1.0">

Given these two types of input, it is possible to automatically convert from one format to the other.

3.2.4 Sound files:<src>

The <src> allows the playing of arbitrary sound files in the middle of documents. This is similar to the equivalent <src> command in HTML for loading images. The SSML definition states that the sound files will be played between the words between which it occurs.

3.3 Multi-Linguality

SSML is not intended to be specific to any particular language and hence is suitable for multi-lingual synthesis. Care was taken in the design of the SSML tags to avoid features which are too language specific. Tags such as <define> are quite general in that they cover any type of phonology - lexical specification of tone for languages such as Mandarin does not require special treatment as <define> allows any type of pronunciation specification. The prosodic control features, <phrase> and <emphasis> may pose more problems. While it is true that all languages have word level emphasis, phrasing and intonational tunes, these may be structured in a different way such that a combination of the three is not sufficient to produce reasonable prosodic control.

These difficulties have not arisen with the small number of languages which have been accommodated within the current SSML framework (English, Spanish and Welsh), but problems may arise with other languages. If such problems do arise, two paths are open: either the prosodic controls may be made more general giving rise to a single set of tags which covers all the languages, or, language specific tags may be adopted where necessary.

SSML allows the language being spoken to change within a document. This is done with

the <language> command:

<language= *language*>, e.g. <language= "english"> <language= "french">.

Synthesizers will differ in what languages they support. When an SSML document requests a language not available to that synthesizer, the subsequent behaviour is system specific: either the text can be ignored or an attempt can be made to speak the text in a different language.

3.4 Controlling the Amount of Specification

A problem that arose in the early development of SSML concerned the amount of specification that a user should be expected to provide. While it is true that a major motivation for using SSML is that it allows more exact prosodic control than can be expected from automatic analysis, TTS systems can still produce some sort of prosodic analysis from raw text, by using punctuation, syntactic analysis (Allen et al., 1987) or part-of-speech tag analysis (Sanders and Taylor, 1995). Testing of early versions of SSML showed that although it was often highly advantageous to make use of the <phrase> and <emph> tags, it was overly cumbersome to have to specify these for every phrase in every document: it was clear that a certain amount of automatic prosodic analysis would be very useful.

The problem occurs with deciding when the system should expect explicit tags to indicate prosody and when it should generate its own. This problem is made more complex by the fact that different TTS systems vary in their ability to produce prosodic markers automatically, and therefore a document with sparse prosodic specification may sound acceptable on one machine and not on another.

Several approaches can be taken to this problem. In the CSTR MTS system, sentence boundaries can be detected with high (but not 100%) accuracy. The view was taken that if no <phrase> markers are placed within a sentence, then that sentence should have all its phrasing determined automatically. If a sentence contains a <phrase> tag, then the sentence boundaries and the <phrase> tags are used to split the sentence into intonational phrases. In these cases, minor punctuation is ignored.

Once this is done a similar approach can be taken for <emph> placement: any phrase (whether explicitly generated by <phrase> tags or automatically) not containing an <emph> tag uses rules to provide a default nuclear accent placement.

A concept which has been examined, but which has not yet been incorporated into SSML is the notion of *specification level*. One can often tell in advance what level of prosodic specification

will be provided. For instance, in applications where the SSML is generated by machine, all the prosodic information will usually be specified explicitly as tags. At the other extreme, sometimes no tagging information will be available at all, and in these cases the MTS system needs to function like a TTS system. To provide for ease of processing, the <level> tag was developed, whereby <level = 1> indicates full specification, <level = 2> indicates the hybrid mode described above and <level = 3> indicates full TTS behaviour.

4 The Edinburgh MTS system

This section discusses the operation of the Edinburgh markup-to-speech (MTS) system (Isard, 1995) and serves as a guideline for how to construct or adapt a synthesis system to use SSML. Figure 2 gives an overview of the document-to-speech conversion process.

4.1 SGML Parsing and Validation

The first stage in the system is the SGML parser. This module takes a document and the SSML DTD and produces a structured output. The parser can operate either with validation mode, whereby illegal SSML constructs are reported as errors and the system halts, or without this mode, whereby the system continues regardless of errors in the hope that some useful output can still be obtained. We have found that the validation mode is useful when preparing SSML documents that one wishes to keep, whereas this mode is best left turned off in when the system is being used "live".

In addition to validation, the parser also performs the useful task of *tag expansion*. Elements are defined as having start-tags (such as <phrase>) and end-tags (such as </phrase>. In many cases, it is possible to define a default behaviour to cut down on the number of end-tags a user has to specify. For example, the most explicit way to mark phrases is as follows:

<phrase>This is the first phrase</phrase><phrase> and this is the second.</phrase>

The DTD specifies that the end phrase tag is optional, and so this example could have been written as:

<phrase>This is the first phrase<phrase> and this is the second.

This is legal because the start of one phrase is defined as also being the end of the previous one. This capability cuts down the number of tags which the user must specify. The parser fills in the missing tags automatically so that the interpreter need not worry about whether optional

tags have been used or not.

4.2 SSML Interpreter

The SGML parser simply expands, validates and tokenizes the input document, according to the specification in the DTD. There is nothing specific to speech synthesis in this process, and the parser should operate in the same way for all implementations. It is the job of the SSML *interpreter* to take this cleaned-up input and interface with the internals of the speech synthesis system. The interpreter is therefore specific to individual synthesis systems.

The Edinburgh MTS system uses a stream based internal data representation formally equivalent to that described in Black and Taylor (1994b). This is a stream based architecture in which linguistic types such as words and phonemes are represented in lists, creating a word list and a phoneme list. Items in different lists may be linked, so that the constituent phonemes of a word may be specified. This representation is similar to that developed by Hertz (1990), although the method of linking items differs significantly.

The function of the interpreter is to fill in basic information in the architecture structure allowing subsequent modules to perform their function. Next, the default rules discussed in section 3.4 are invoked to ensure full prosodic specification. The <define> function works somewhat differently in that word pronunciations defined in this way are kept in a special lexicon, which is consulted before the main system lexicon. The position in the document where the definition occurs is noted so that only words following a given <define> command can make use of it.

4.3 Synthesizer Operation

Once the architecture structure has been filled using the SSML tags and default rules, the generation of speech proceeds in a manner similar to normal synthesis systems. This part of the process is obviously highly specific to the workings of the individual synthesis system and need not be described in detail here. However, it may prove useful to give an example of how features are actually implemented to aid other developers wishing to create a MTS system.

The Edinburgh MTS system uses an autosegmental representation of intonation based on the concept of *intonational events*, a generic term used for pitch accents and boundary tones (Taylor and Black, 1994). An event stream is used within the architecture to store intonational event information. When a word occurs with emphasis marked in the document, an intonational

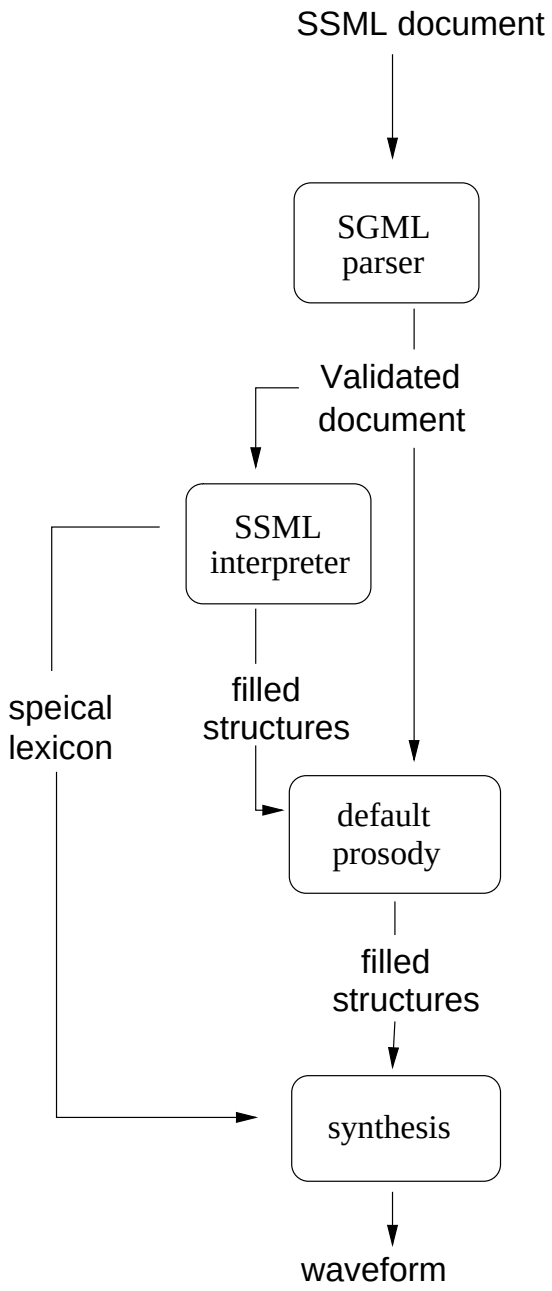

Figure 2: Processing SSML documents in the Edinburgh MTS system

event is placed in the event stream and this event is then linked to the word. Depending on the tune of the current phrase, boundary tones may also be placed in the event stream. Intonational rules then use the tune information to decide what the type of the pitch accent and boundary tones should be. Using the lexicon and/or letter to sound rules, a pronunciation for each word is determined. This process fills the syllable (with stress information) and phoneme streams. The pitch accent is then linked to the stressed syllable of the emphasised word, which is important for determining the exact position of the pitch accent. An algorithm constructs a set of pitch targets from the information in the event stream and a subsequent process converts these into an F0 contour.

5 Applications

5.1 Filters

An obvious question concerning the design of SSML is whether it is necessary to have a distinct markup language for speech synthesis, rather than simply building a speech synthesis interpreter for document markup languages such as HTML and Latex. Several attempts have been made to develop such systems, the most notable being the Aster system developed by Raman (1994), which provides a comprehensive Latex-to-speech conversion facility.

It is obviously of great benefit to be able to synthesise documents that have already been marked up in Latex or HTML. However, we believe that the design of a HTML-to-speech system is probably not the best solution to this problem. First of all, an interpreter would have to be built for each of the different document markup schemes in existence and this would have to be performed for every individual MTS system. Secondly, this would disallow the use of tags that are really only relevant to speech synthesis, such as <phrase>. Our solution has been to advocate the design of markup language *filters* which convert from HTML or Latex to SSML. Such filters are relatively easy to build and save MTS developers from having to worry about the interpretation of more than one markup language. Markup languages based on SGML, such as HTML, are particularly easy to design filters for.

A system such as Aster could be adapted so as to convert Latex documents into SSML rather than the Dectalk commands it currently uses. In that way, it would be possible to use Aster on a variety of synthesis platforms.

5.2 Machine Generation

While most HTML or latex documents are physically typed at keyboards, it is envisaged that a major use for SSML will be in the speech interface in language generation systems, including spoken dialogue and machine translation systems.

It is widely accepted that text is not an intelligent interface format for such systems. Since a machine has generated the utterance in the first place, it will know a substantial amount about the structure of that utterance. It would therefore be wasteful to throw away this information, output the text, and then require a text-to-speech system to parse this text.

Ideally it should be possible to couple the language generation component and the synthesis component together directly by allowing them to share complex data structures. However, our experience of working on machine translation projects has shown that this is problematic as it is difficult for the builders of both the speech synthesis and machine generation component to specify a common format and stick to it. Although it does not allow the intricate level of control that a tightly integrated language-generation/speech-synthesis component system would allow, SSML does provide a stable and powerful common interface which frees the developers of both systems from intricate interfacing issues. Equally importantly, the language generation component may be coupled with a different SSML-compatible synthesis if desired.

The term "language generation" used in this context need not denote a sophisticated linguistic system, any sort of message generated by a computer can be used to produce SSML documents. For instance, we have developed a C function called *ssmlprintf()* which takes strings as arguments in much the same way as the standard C *printf()* function, but instead of printing these strings, sends them to a speech synthesis daemon, which synthesises the message.

6 Future Additions to SSML

This paper has concentrated on describing the features of SSML version 1.0, which is the version currently in use in the Edinburgh MTS system. During the development and subsequent use of this version, many recommendations have been made as to how the version 2.0 version should behave, which are discussed below.

6.1 Style Files

It is not always easy to partition the functionality of a document processing system into logical and physical aspects as there are areas of the formating process which do not fall neatly into either of these categories. Such problems also exist in speech synthesis markup.

A case in point concerns control of speaking style in SSML. By default this is left to the MTS system as the input documents are not meant to contain any explicit instructions as to how the utterances are to be spoken. This requirement is necessary to preserve the platform independence of the language. In most circumstances it is not beneficial for the author of a document to be able to specify stylistic matters: in this way a MTS system can process documents from many sources with the knowledge that the output can be spoken in a uniform style.

However, there are circumstances when it can be argued that a certain amount of style control is genuinely part of the logical content of the document. For example the author of a document may wish to simulate a conversation between two people, with the obvious requirement that each should have a different sounding voice. It would be a gross violation of the SSML design to allow direct choice of voice in documents, but an acceptable compromise involves the use of user supplied *style files*. In this scheme, the document contains tags such as <style voice = *speaker1*> which tells the system to speak with a user defined set of speaker specific parameters called *speaker1* until the next <style voice> tags is reached. A <style voice> tag with no associated attribute resets the MTS to use the default voice.

The author also supplies a style file which gives information about this voice. These specifications are not mandatory in the sense that if the system cannot provide a voice of this nature the default voice will be used. The definition of the voice type is given as a set of features which allows approximate matching of voice type if an exact match is impossible. It is possible that types of voices may become standard enough in the future to be given universally accepted names in the same way as fonts are currently named.

Other types of style definition are also of interest potentially, for example it may be important to the function of a discourse that a particular stretch of speech is spoken quickly. Again, user specific style declarations may be invoked e.g. <style manner=*manner1*>, which again relates to a specification in the user supplied style file relating *manner* to an instruction to speak quickly. Exactly which tags should be allowed in style files is still an open question. It is probable that these tags will be kept distinct from SSML proper and be defined in a standard library.

6.2 Mathematical Markup

SSML will eventually include a mathematical formula handling ability. For sake of simplicity and standardisation, we have decided to make the math markup system the same as that used in the forthcoming HTML 3.0. Since the math DTD for HTML 3.0 has not yet been finished at time of writing, it has not been possible to fully integrate a math mode into SSML.

The Aster system (Raman, 1994) provides a thorough specification of how mathematical formulae marked-up in Latex and Tex should be spoken. The HTML draft math syntax is based quite closely in the Tex formatting language, and so it should be relatively straightforward to adapt Aster to read HTML math format.

6.3 Evolution and Standardisation of SSML

This paper has presented SSML version 1.0, which amounts to a prototype speech synthesis markup language. While considerable effort has been expended on designing SSML to achieve its stated goals of convenient synthesizer control and platform independence, it is clear that as SSML is used more and more, changes and adaptations will be required. In particular investigation of cross natural language suitability will only be possible after SSML compatible synthesizers have been implemented for a number of languages. We hope that SSML will gain wider usage and in so doing will evolve into a standard that will help the integration of speech synthesizers into other systems.

Appendix: Example SSML Documents

In example A the position of the phrase boundary determines the meaning of the sentence. The phrase boundary information could not be deduced from the plain text without a great deal of real world knowledge.

Example A

```
<ssml>
  <phrase>I saw the man in the park<phrase> with the telescope</phrase>
  <phrase>I saw the man<phrase>in the park with the telescope</phrase>
</ssml>
```

Example B shows how a document may contain more than one language. In this case, the default language has been set to English.

Example B

```
<ssml>
  <phrase>This sentence begins in English</phrase>
  <language="italian">
  <phrase>continua in Italiano</phrase>
  <language="german">
  <phrase>und endet auf Deutsch</phrase>
</ssml>
```

This example demonstrates the facility in SSML which allows word pronunciations to be defined and redefined throughout a document, with an SSML version of the well known song which contrasts English and American pronunciations. The lexicon used is by default the (British English) CSTR lexicon.

Example C

```
<ssml>
  <phrase> you say either
  <phrase> <define word="either" pro="ii1 dh @">  and I say either
  <phrase> you say tomato
  <phrase> <define word="tomato" pro="t @ m ei1 t ou"> and I say tomato
</ssml>
```

An alternative way of achieving the same output is show in example D.

Example D

```
<ssml>
  <define word="either1" pro="ii1 dh @">
  <define word="tomato_br" pro="t @ m aa t ou">
  <define word="either2" pro="ai1 dh @">
```

```
  <define word="tomato_usa" pro="t @ m ei1 t ou">

  <phrase> you say either1 <phrase> and I say either2
  <phrase> you say tomato_br <phrase> and I say tomato_usa
</ssml>
```

References

Allen, J., Hunnicut, S., and Klatt, D. (1987). *From Text to Speech: the MITalk System*. Cambridge University Press.

Black, A. W. and Taylor, P. A. (1994a). Assigning intonation elements and prosodic phrasing for English speech synthesis from high level linguistic input. In *ICSLP '94, Yokohama, Japan*.

Black, A. W. and Taylor, P. A. (1994b). CHATR: A generic speech synthesis system. In *COLING '94, Kyoto, Japan*.

Crystal, D. (1969). *Prosodic Systems and Intonation in English*. Cambridge Studies in Linguistics. Cambridge Universtity Press.

Goldfarb, C. F., Mosher, E. J., and Peterson, T. I. (1970). An online system for integrated text processing. *Proceedings of the American Society for Information Science*, 7:147–150.

Hertz, S. R. (1990). The delta programming language: an integrated approach to non-linear phonology, phonetics and speech synthesis. In Kingston, J. and Beckman, M. E., editors, *Papers in Laboratory Phonology 1*. Cambridge University Press.

Isard, A. C. (1995). *SSML: A Markup Language for Speech Synthesis*. PhD thesis, University of Edinburgh.

Ladd, D. R. (1983). Phonological features of intonation peaks. *Language*, 59:721–759.

O'Connor, J. D. and Arnold, G. F. (1973). *Intonation of Colloquial English*. Longman, 2 edition.

Pierrehumbert, J. B. (1980). *The Phonology and Phonetics of English Intonation*. PhD thesis, MIT. Published by Indiana University Linguistics Club.

Raman, T. V. (1994). *Audion System for Technical Readings*. PhD thesis, Cornell University.

Sanders, E. and Taylor, P. A. (1995). Using statistical models to predict phrase boundaries for speech synthesis. In *Proc. Eurospeech '95, Madrid*.

Silverman, K., Piterelli, J., Beckman, M., Pierrehumbert, J., Ladd, R., Wightman, C., Ostendorf, M., and Hirschberg, J. (1992). Tones and break indices: a standard for prosodic transcription. In *International Conference on Speech and Language Processing '92, Banff, Canada*.

Taylor, P. A. (1992). *A Phonetic Model of English Intonation*. PhD thesis, University of Edinburgh. Published by Indiana University Linguistics Club.

Taylor, P. A. and Black, A. W. (1994). Synthesizing conversational intonation from a linguistically rich input. In *Second ESCA/IEEE Workshop on Speech Synthesis, New York, U.S.A.*