SSML: A Markup Language for Speech Synthesis

Amy Isard

MSc Information Technology: Knowledge Based Systems

Department of Artificial Intelligence

University of Edinburgh

1995

Abstract

At present, most speech synthesis systems use unconstrained text as their input, which is desirable from a human point of view, but problematic as far as machines are concerned, since the process of converting text to speech is very complex.

In this thesis, the development of a Speech Synthesis Markup Language (SSML) is described. The SSML format is a compromise between human and machine needs. SSML is an application of the Standard Generalized Markup Langauge (SGML). The input is text-based and unconstrained in its use of words, but allows a large amount of extra information to be included with the text to guide the synthesis in its performance. The text is annotated with markers to specify features such as emphasis, particular speech styles, or the beginning of new topics.

A prototype version of SSML is described, with the Markup-to-Speech program written for use with the CSTR speech synthesizer. A larger set of markers for a future version of SSML is also presented.

Acknowledgements

I would like to thank my first supervisor, Chris Mellish, for all his help and encouragement. I would like to thank my second supervisor, Paul "God" Taylor, for his extensive advice and his tireless willingness to be interrupted at any time.

I would also like to thank my various flatmates and "extra inhabitants" throughout the year for keeping me sane and cheerful, particularly Simon to whom I now owe encyclopedias.

This course was funded by an EPSRC studentship.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

Scientists first began trying to recreate human speech many centuries ago, and the first attempts used models of the human vocal tract apparatus through which air was propelled by mechanical means. The first real attempt at understanding and reproducing the generation of sounds in the vocal tract was made by Wolfgang Von Kempelen of Vienna in 1791 [Fla84] [Ste85]. Over the next century and a half, more and more sophisticated versions of this sort of machine were built, but it was not until the twentieth century, with the advent of complex electronic devices, that speech synthesis really took off. An important development was Homer Dudley's VODER (Voice Operation Demonstrator) demonstrated at the World Fair in New York in 1931. It used electrical networks whose resonances were similar to those of individual speech sounds, and required an operator with up to a year's training to "play" its keys to produce intelligible speech [Fla84] [Ste85].

Today, almost all speech synthesis is carried out entirely by computer (for a summary of the techniques used, see chapter 3). Some applications of speech synthesis (for example reading machines) involve taking large amounts of text as input. A system which takes ordinary written text as input and outputs speech is known as a Text-to-Speech system. This is usually abbreviated to TTS, and this abbreviation will be used throughout this thesis.

1.2 Problems of TTS

The use of text as input for a speech system is attractive from the point of view of human users, as it means that no alterations have to be made to the text to convert it from a form which can be read by a human to one which can be read by the machine. However, text is not the ideal input from a machine point of view, as the process of converting text to speech is complicated. In a TTS system, the text must pass through numerous stages before it is ready to be synthesized. This process is computationally expensive and also prone to error, as human-readable text is both ambiguous and under-specified in terms of linguistic information. TTS systems employ various methods to try to cirumvent this problem; however, these methods are are all system-specific, so no standard solutions have emerged. The Speech Synthesis Markup Language (SSML) provides a standard method for dealing with the sort of difficulties outlined above.

1.3 Aims of SSML

In an SSML document, markers are added to the text which contain extra information which is not made explicit by the text itself. SSML is implemented as an application of the Standard Generalized Markup Language (SGML) described in chapter 4. A set of rules are defined which specify what can make up an SSML document, and documents can then be written which conform to these rules. In the version of SSML presented in this thesis, it is possible, for example, to mark the beginning and end of each phrase, and to explicitly mark which word in a sentence should be emphasized. Some of the information included in SSML tags could be extracted from a text using traditional TTS methods, but SSML also allows the inclusion of tags which specify qualities such as the pitch or volume of the speech, or the mood in which the text is to be spoken.

1.4 Structure of the Thesis

In this thesis I present two aspects of SSML:

- A prototype version of SSML. Documents written in this style can be spoken by the CSTR Speech Synthesizer (see section 6.1) once they have been processed by the Markup-to-Speech (MTS) program implemented as part of this project and described in detail in chapter 6,

- An extended structure for an expanded version of SSML, described in chapter 7.

Chapter 2 contains an introduction to typical Speech Synthesis Systems and their drawbacks, along with the solutions proposed by SSML. Chapter 3 gives an overview of methods currently used in speech synthesis. In chapter 4, I provide an introduction to SGML. Chapter 5 contains a detailed description of the prototype version of SSML developed for this project. Chapter 6 provides a description of the conversion of text in SSML format to a form suitable for a synthesizer. Chapter 7 contains a discussion of more complicated markup issues, and of possible directions which a more elaborate SSML system might take. The thesis is then concluded by an evaluation of the project as a whole.

Chapter 2

Speech Synthesis Systems

The term *speech synthesis system* will be used here to describe any system which takes some form of high level input, transforms it into input suitable for a speech synthesizer, and then performs the synthesis. It will henceforth be abbreviated to SS system. An SS system can be thought of as comprising two stages, *analysis* of the text (in whatever format it is presented) and then *synthesis* of the speech. In this chapter, I will describe different approaches to the analysis stage. A description of speech synthesis techniques is contained in chapter 3. Within the analysis section in this chapter, I will first describe SS systems in general terms, and then describe different approaches to the analysis of the text. I will first focus on TTS systems and their drawbacks, and then briefly describe some systems which do not take ordinary human-readable text as input. This will be followed by a description of SSML solutions to the problems described.

2.1 Applications of SS Systems

There is a great variety of different tasks to which speech synthesis systems can be applied [Ste85] [Kla87], and a selection is listed in figure 2–1.

The form of the input to a system will depend on the application for which the particular synthesis system is intended. Some will take as input unrestricted

- Reading machines for the blind
- Speaking aids for the handicapped
- Talking computer terminals
- Remote access to electronic mail
- Teaching machines and training aids
- Talking books to teach reading
- Flexible database enquiry systems
- Remote access to information over the phone
- Talking instrument panels and warning systems
- Hobby computers
- Proofreading

Figure 2–1: Commercial SS Applications

text such as would be found in a printed text (reading machines for the blind and proofreading for example), whereas others could more usefully accept text in other formats.

2.2 Analysis

2.2.1 Overview of TTS Systems

Text-to-Speech systems aim to mimic the performance of a human reading aloud from a printed text. If the text to be read is not already on-line, an optical character recognition system which translates optical images into computer-readable text may be attached to the TTS, creating a reading machine. The major commercial application for TTS systems is currently systems which can act as an

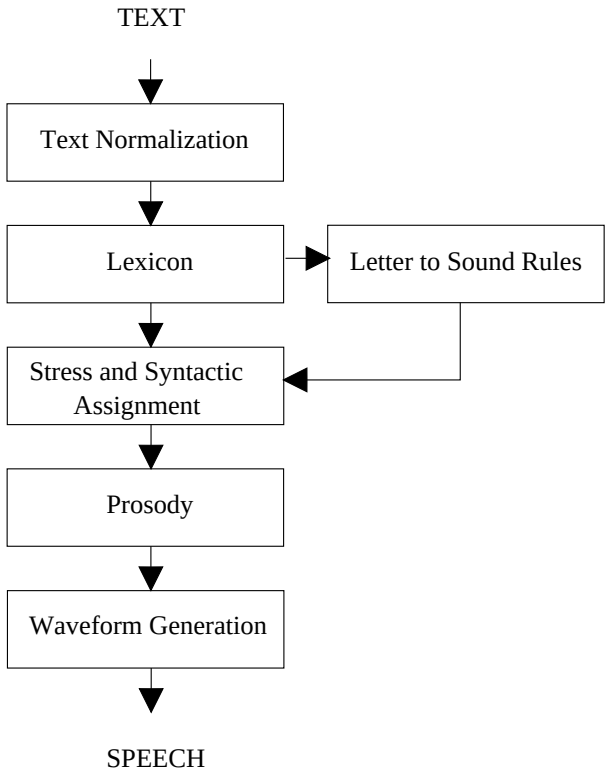

Figure 2–2: Modules of a Typical TTS System

aid to vocally and/or manually handicapped people. Apart from this, TTS systems are also used in phone-line information systems such as weather forecasts or long-distance personal email readers, and as proofreading aids [Tra95].

To perform TTS, it is necessary to undertake a detailed linguistic analysis of a text, in order to determine parameters which will make it possible to produce natural sounding speech. A typical TTS will have several modules connected in a serial manner, as in figure 2–2.

First, the text must be converted into a standard form. This means for example that abbreviations and acronyms must be expanded, and numbers must be written out in full.

Then, the text must be converted into phonetic symbols. This can be done

either by using a dictionary, or by using a set of rules which convert combinations of characters into sounds, or by a combination of the two.

After this, information about the syntactic structure of the text is added, along with information on which words and syllables will be stressed.

Lastly the prosody (intonation and rhythm) of the phrase is determined, and a waveform is then generated.

2.2.2 Common Problem with TTS Systems

There are several points at which TTS systems commonly have difficulties. These include:

- Conversion of symbols to standard form,
- Correct pronunciation of foreign or unusual words,
- Correct assignment of prosody and stress.

TTS systems typically either attempt to ignore these problems, or invent ways of getting round them which are specific to their particular system.

The ways in which two commercially available systems address some of these problems will now be presented.

In the MITalk system [AHK87], text is initially preprocessed to remove any symbols which will not be accepted by *FORMAT*, the first module proper of the MITalk system. MITalk has a library of special characters which translate into commands to the synthesizer at a later stage. For example, when a whitespace is found at the beginning of a line, this is assumed to be the beginning of a new paragraph, and a "." is added to the text, which will later be translated into a pause in the speech. When a "." is found in the text, it is treated as an end of sentence marker unless the word preceding it is found in a table of abbreviations,

in which case the abbreviation is expanded. Words which contain only capitals, or which contain digits as well as letters, are considered to be symbols, and are translated by pronouncing each character separately.

Another commercially available system is the TrueTalk TTS from Entropic, which is based on software developed at AT&T Bell Labs.[1] In the TrueTalk TTS, the user is allowed to alter certain parameters of the speech.

The default TrueTalk pronunciation of words may be overridden by including phonetic transcriptions in the input text, or by composing a file which contains a set of phonetic transcriptions. The inclusion of transcriptions in the text is done by means of escape sequences. If a file of pronunciations is used, it is written in the same format as the normal TrueTalk dictionary, and is consulted before this dictionary.

There is also a mechanism for "filtering" input text before it is sent to the TTS engine, which allows the user to specify certain parameters. The two filters available are a speech-enlivening filter and an email filter. The enlivening filter inserts prosody control parameters for the TTS engine in the input text stream, to make the speech sound more animated or spirited. The email filter assumes that the input text contains structures commonly found in email, such as "To", "From" and "Subject" lines, and tries to translate the text into an intelligible form, including the information contained in the email header. This is done using escape sequences which are inserted into the text before it is sent to the TTS engine.

The solutions used by MITalk do not allow the user to make any changes to parameters, and will be largely successful in practice, generating only a few errors, but they are not standard in any way, and are particular to the MITalk

[1] All the information on this system was obtained from the Entropic www pages [Ent]. (www addresses valid as of 11 September 1995.)

system. TrueTalk does allow the user some flexibility, but again, the solutions are completely system-specific.

2.2.3 Other Speech Synthesis Systems

In systems which produce synthesized speech, there will not necessarily be a text stage. If the input is from a natural language generation system, there will already be a great deal of syntactic information which can be used in improving the quality of the synthesis [GMS94]. Systems which function in this way are known as Concept-to-Speech systems, and the speech is generated directly from an abstract representation. Other systems use different forms of text as input. A system has for example been implemented using Bliss symbols[2] as input [CG86]. T.V. Raman's AsTeR [Ram91] is a system which performs audio formatting on electronic documents written in LaTeX to produce intelligible speech. It can process both literary texts and documents containing complex mathematical formulae.

2.2.4 SSML Solutions

SSML provides a flexible interface which can be used as a module of any system whose end aim is the synthesis of speech. As it is a completely specified language, it is possible to write a program which will transform any other form of input into SSML format. It can be used with any speech synthesizer, and will therefore provide a standard interface which will carry out part of the transformation between a text of any format and a speech synthesizer of any design. This will naturally mean that technologies developed by different research groups will become more compatible with one another, and it will be easier to integrate systems.

[2]The Bliss symbol notation system is a shorthand notation for words in a text

AsTeR and the Bliss system use totally different techniques from one another, and do not pass through any common stages, but SSML could provide a module common to both.

SSML would also handle all the problematic processes described in the two commercial TTS systems in a standard manner:

- The beginning and end of paragraphs and sentences is explicitly included in the text using start- and end-tags (see chapters 4 and 5 for details).

- It is possible to change the mood or style of speech in as many ways as the available synthesizer technology provides.

- A section may be marked as one which contains email information and will be treated accordingly.

- The pronunciation of a word may be included directly in the text, using a "pronunciation" tag. A style sheet (see section 7.4) may be attached to SSML which includes information which is specific to a particular speech synthesis application. For example, a list of common abbreviations can be included in a style sheet, and every time one of these is encountered in the text, it will be automatically expanded.

- If multiple speaking voices are required, this could also be easily specified using SSML. The voices required are specified in the style sheet, and each section of text is then marked as being spoken by a particular speaker.

On its own, SSML is not sufficient to perform synthesis, but a conversion program incorporating a standard parser (see chapters 4 and 5) can easily be written for a given synthesizer. One such program written as part of this project for the CSTR Speech Synthesizer is described in detail in chapter 6.

Chapter 3

Synthesis

In this chapter, I will briefly describe current techniques used in speech synthesis.

3.1 Definitions

To describe approaches to speech synthesis, it will be necessary to introduce certain terms:

Phoneme The phoneme is the smallest unit of speech which distinguishes one word from another within a particular language. Phonemes can be identified using *minimal pairs*, which are pairs of words which have different meanings, but differ only in one phoneme. In English, "sat" and "pat" are a minimal pair, as are "pat" and "pet". This tells us that the "s" and "p" sounds are both phonemes, as are the "a" and "e".

Phone A phone is one particular realization of a phoneme in an utterance.

Allophone Several phones may belong to the same phoneme within a language, and these phones are then known as allophones of the phoneme. For example, in English, there are two allophones of the phoneme /l/. "Clear /l/" is used

in syllable-initial position as in "loud". "Dark /l/" appears in syllable-final position as in "call". Since there is no minimal pair for which these two sounds make a distinction, they cannot be separate phonemes, and must therefore be allophones.

Diphone A diphone is defined in speech synthesis as half of one phone followed by half of the next.

Formant When a speech signal is analyzed into a signal spectrum, peaks appear at certain frequencies corresponding to vocal tract resonances, and these are known as formants.

3.2 Current Methods of Speech Synthesis

The various approaches to speech output from computers can be grouped into the following categories [Tra95]:

1. Articulatory Synthesis,

2. Synthesis by rule,

3. Concatenative Models.

3.2.1 Articulatory Synthesis

In this approach, the propagation of sound through the vocal tract is modelled using physiological and mechanical information about the shape of the vocal tract, the activity of the vocal chords, and the movement of the lips and tongue [Ste85]. There is a physical model of the human vocal tract, and a set of rules which maps a sequence of speech units on to movements of the articulators. This method is

attractive because of the clear way in which it is related to human speech production, and synthesis carried out in this way can be of very high quality. However, it is very difficult to accomplish successfully, and a great deal of computational power is required. It is also difficult to acquire the necessary data, as subjects must be x-rayed while speaking.

3.2.2 Synthesis by rule

In this approach, real speech data are analyzed, and a set of rules is established which allow a waveform to be created from a series of allophones. Formant Synthesis [Fla84] is the most common of these methods.

3.2.3 Concatenation of Speech Units

Any size of unit may be chosen when this approach is used, and some possibilities, in decreasing order of size, are:

- Whole Phrase
- Part of Phrase
- Word
- Syllable
- Diphone
- Phoneme

The larger the unit used, the more storage space will be required; to get a decent sized vocabulary in English, you would need at least 2500 words, but only about 40 phonemes. It will obviously never be possible to create a database of all the possible words in a language, as the number is constantly growing, and in any

case, it would be more useful to have the capability to produce any nonsense word if desired, and this is not an option if a dictionary of words is being used.

Once the units have been selected, the problem of joining them together in a natural-sounding way must be addressed. This creates a problem if phonemes are chosen, as the articulation of a given phoneme depends heavily on its context, i.e. the phonemes which come before and after it. The unit most commonly used in speech synthesis is the diphone [IM86]. Rather than an example of each phoneme being stored, an example of a transition between every possible pair of phonemes is stored. An advantage of diphone synthesis is that the middle section of a phoneme is generally the most stable part, so that when two diphones are concatenated, a fairly clean join is made. A problem of concatenative synthesis in general is that it is difficult to change the prosody of an utterance. This requires that modifications be made out to the duration, pitch and amplitude of parts of the speech signal while the spectrum of the signal remains the same. The most common solution to this problem for diphones is the use of PSOLA synthesis [CM89].

Chapter 4

Standard Generalized Markup Language

In this chapter, I shall provide an introductory and somewhat simplified description of the Standard Generalized Markup Language (SGML), in order to give an overall impression of its structure and uses. I will generally only include information on the features of SGML which were used in defining SSML. There are of course numerous books on the subject, written in different styles and with different groups of people in mind; for further details consult Appendix A.

4.1 History and Rationale of SGML

When it is said that a text has been marked up, what is meant is that labels have been added to it which give information about its structure. A markup language is a set of descriptions which express how text should be processed, or handled in other ways. Generalized markup is special in that it does not specify *how* this processing should be carried out. For example, if you were preparing a text for printing, and wanted to have section headings, rather than specifying that chapter titles should be written in 14 point bold Helvetica, and section headings in 12 point italic Times, you would mark up the first heading as "chapter" and the second

as "section" and leave it to subsequent processing to decide what the effect of this would be. This means that if you wanted to send the same document to two different conferences, say, and they had different ideas about how they wanted the finished product to look, it would not be necessary to go through changing the headings one by one, as you could simply change the specification of which font is used when a chapter heading is found. This may sound familiar, as this is much the way LaTeX works, but LaTeX is not an SGML application; it was developed from the TeX system, separately from SGML, so uses a different syntax for start- and end-tags, and it also allows the user to specify non-structural information. Once a document has been marked up with SGML, it may be output in any way the user desires, and is portable between systems.

The philosophy of SGML is neatly summarised in this quote from [Gol90]:

> "Markup should describe a document's structure and other attributes rather than specify processing to be performed on it, as descriptive markup need be done only once and will suffice for all future processing."

The term markup as it is used in the context of markup languages comes originally from the publishing and printing business, where it means the instructions for the typesetter that were written on a typescript or manuscript copy by an editor. The first precursor of SGML was GML, invented by Charles Goldfarb, Edward Mosher and Raymond Lorie (G, M and L respectively) in 1969 as part of an IBM research project on integrated law office information systems, as a means of allowing different components of the system to share documents. Goldfarb continued research on document structures, and over the next 15 years, a test description language standard based on GML was developed. The Standard Gen-

eralized Markup Language became an international standard for the description of marked-up electronic text in 1986.[1]

4.2 Basic SGML

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 section 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. SGML has other capabilities which will not be described in detail here (full descriptions can be found in the sources listed in Appendix A). One is the definition of *Entities* in SGML; one use of entity sets is to define foreign character sets when documents are being prepared for printing, and standard public sets are available. These definitions can be included either in the separate DTD file or at the beginning of a document instance. The part of the document containing the DTD and the entity declarations is known as the *prolog*.

---

[1]It is defined in an International Standard published by the International Organization for Standardization (ISO), with reference number ISO 8879:1986, bearing the full name "Information processing – Text and office systems – Standard Generalized Markup Language (SGML)". See the SGML www pages [SG] for information on how to obtain a copy. (www addresses valid as of 11 September 1995.)

```
<!element A - - (B+ , C)*>
<!element B - - (#PCDATA)>
<!element C - o (#PCDATA)>

<!attlist B size (small | medium | large) medium>
```

Figure 4–1: SGML DTD

4.3 The DTD

The DTD also comes in two parts:

1. the names and combination rules for the elements,

2. 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 List Declaration.*

An example DTD is included in figure 4–1 so that it can be used for reference while an explanation of each part of the DTD is given below.

4.3.1 Element Declaration

In this part of the DTD, the only way that an element is defined is in reference to how it may be combined with other elements. The definition comes in three parts:

1. the name of the element,

2. rules about start- and end-tags,

3. rules about combining this element with others.

The name of an element is its *Generic Identifier*. The tag rules are the *Minimization Rules*. The element combination rules are known as the *Content Model*.

Generic Identifier

Each element is given a unique name within the particular DTD. In figure 4–1, there are three elements, named A, B and C.

Minimization Rules

The second part of the declaration specifies minimization rules for the element concerned. The beginning of an element in a document instance (described below in section 4.4) is normally signified by a *start-tag*, and the end is specified by an *end-tag*. The minimizationn rules determine whether or not start- and end-tags must be present in every occurrence of the element concerned. They take the form of a pair of characters, separated by whitespace, the first of which relates to the start-tag and the second to the end-tag. In both cases, a hyphen or a letter "o" (for omissible or optional) is given; the hyphen indicates that the tag must be present, and the "o" that it may be ommitted.

In figure 4–1, both start- and end-tags must be present for elements A and B, but the end-tag for element C may be omitted.

Content Model

In this part of the element description, the way in which elements may be combined is described in terms of *Connectors* and *Occurrence Indicators*. In addition to the element names, it is also possible to specify that text may be entered as the body of an element; this is done using the string #PCDATA.

Connectors There are three connectors: | , & which have their standard logical definitions.

(A | B) means that either element A or element B may be included, but not both.

(A , B) means that element A and element B must both be included, in the order in which they appear in the declaration.

(A & B) means that element A and element B must both be included, but may occur in any order.

Occurrence Indicators There are three occurrence indicators: ? * + which also have their standard definitions.

A? means that there may be one occurrence of A, or none at all.

A* means that there may be any number of occurrences of A, or none at all.

A+ means that there must be one or more occurrences of A.

Occurrence indicators can be applied to elements which have been joined by connectors, as well as to single elements.

When a document is prepared according to the DTD in figure 4–1, it will contain one element A, which in turn must contain the combination (at least one B, followed by one C) any number of times. Elements B and C contain unrestricted text.

4.3.2 Attribute List

The attribute list for an element contains information which will be included in its start-tag. An element may have any number of attributes, each one of which will

```
<!doctype A system "short.dtd"
[]>
<a>
  <b size="small">some text</b>
  <b>some more text</b>
  <c>even more text
</a>
```

Figure 4–2: SGML Document

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 it may be possible to use unconstrained text to specify the value. It is also possible to set a default value, in case none is specified. The name of the element comes first, and then the name of the attribute. This is followed by the list of possible values, or #PCDATA, and in the case where a set of values has been specified, one of these may be repeated at the end, and will then act as the default.

In the DTD in figure 4–1, only element B has an attribute list. There is one attribute, size, for which the values may be small, medium or large, and the value is set to default to medium if no value is explicitly set.

4.4 Example Document

Figure 4–2 shows a document which contains the DTD defined in figure 4–1, which for these purposes will be given the filename short.dtd.

There is one A element, started by <a> and ended by . Within this top level element, there is one B element whose size value has been set to small, one b element whose size has not been specified, and will therefore default to medium,

and one C element, whose end-tag has not been included, since it is implied by the A element end-tag at the end.

4.5 Functions of a Parser

A parser[2] is used to process an SGML document. A definition of the minimal function of an SGML parser, as stated in [Gol90], is

> "A program [...] that recognizes markup in SGML documents."

There are two functions which may be carried out by any given parser, these are *validation* and *parsing*. Each parser will perform one or both of these actions.

4.5.1 Validation

There is a formal definition of a validating parser in [Gol90]:

> "A conforming SGML parser that can find and report a reportable markup error if (and only if) one exists."

The most basic type of validating parser will therefore be one which produces a yes/no answer when confronted with an SGML document. More complex versions will give an error message which states the point at which a problem occurs and gives a summary of the error. However, not all parsers will perform validation; some simply parse a document as described below without checking for errors.

[2]There are numerous public domain and commercial parsers available, see the SGML www page [SG] for information on how to obtain them (www addresses valid as of 11 September 1995.)

```
(A
ASIZE TOKEN SMALL
(B
)B
ASIZE TOKEN MEDIUM
(B
-some text
)B
(C
-some more text
)C
)A
C
```

Figure 4–3: Parsed SGML Document

4.5.2 Parsing

At its simplest, parsing consists of producing a new instance of a document in canonical form. The canonical form is a simple text representation of the Element Structure Information Set of the document, which is the information set which an SGML application should act upon. When parsing is carried out on the document in figure 4–2, using the **nsgmls** parser described in section 6.3, the result is as shown in figure 4–3.

This is an acceptable canonical representation of the Element Structure Information Set, but not the only one; other parsers will produce a different format containing the same information. When figure 4–3 is compared with figure 4–2, an obvious correlation can be seen. In the parsed output, the beginning of an element is denoted by an opening parenthesis followed by the name of the element, and the end of an element by a closing parenthesis followed by the name of the element. The attributes of an element are listed before the element, prefixed by an "A"

(for attribute), and with the value listed after "TOKEN". The default attribute value for the second B element, which was not specified explicitly in the document, has been inserted. The end of the C element, which was also not explicit in the document, has also been inserted. The end of the parser's output is marked by a single "C".

Chapter 5

Speech Synthesis Markup Language

This chapter will describe the prototype version of SSML implemented during this project. Eventually SSML will be expanded to include many more options, but due to constraints of time and available synthesizer technology, a subset of the possible options were included at this time. A discussion of an extended version of SSML is included in chapter 7.

5.1 Theory of SSML

Prototype SSML

Four elements were chosen for inclusion in the prototype version of SSML:

1. Phrase Boundaries,
2. Emphasized Words,
3. Specified Pronunciations,
4. Inclusion of other sound files.

The first two of these are implemented by altering parameters of the speech synthesizer, as described in [Tay93], and a summary of the processes is provided in section 6.4.2. The third and fourth options were implemented solely as part of the conversion program written for this project and are described in section 6.4.

Phrase Boundaries

One of the most important things to take into account when trying to produce natural sounding speech is the phrase structure of the text, as this defines the overall intonation contour of a phrase. In an SSML document, the text is explicitly separated into phrases, and although at present there is no means for distinguishing between different types of phrases (eg. question vs statement), this would be an obvious next step to take (see section 7.3).

Emphasized Word

Each phrase may contain an optional emphasized word, and this can greatly aid the naturalness of the speech produced.

Specified Pronunciations

Between phrases, it is possible to redefine the pronunciation of a word. In an extended version of SSML, it would also be possible to include pronunciation specifications in a style sheet (see section 7.4).

Included Files

The user may specify another file from which a sound is to be played.

```
<!element ssml - - ((sound | define)*, phrase,
                    (phrase | sound | define)*)>
<!element phrase - o (#PCDATA | emph)+>
<!element emph - - (#PCDATA)>
<!element sound - o EMPTY>
<!element define - o EMPTY>

<!attlist sound src CDATA #REQUIRED>
<!attlist define word CDATA #REQUIRED
                 phonemes CDATA #REQUIRED>
```

Figure 5–1: SSML DTD

5.2 Structure of SSML

As described in the previous chapter, an SGML document consists of a prolog and a document instance. I will provide SSML examples of both of these separately, and a description of their content.

5.2.1 SSML DTD

Figure 5–1 shows the SSML DTD.

At this point, I will give a purely structural account of the element declarations in the DTD. A functional description is given in the next section.

Element Declarations

There are six element declarations: **ssml, phrase, sound, define** and **emph**.

ssml is the top level element. Both start- and end-tags must be explicitly included. Its content model specifies that there must be at least one **phrase** element, and that apart from this there may be any number of **phrase**, **define** or **sound** elements, in any order.

sound is an empty element, so no text will be included in it.

define is an empty element, as above.

phrase may contain unconstrained text or an **emph** element in any position within the text. Its end-tag need not be specified, as it can be assumed that one **phrase** element has finished when a new **phrase**, **sound** or **define** start-tag appears, or an **ssml** end-tag is encountered.

emph contains unconstrained text, and both start- and end-tags must be explicitly included.

Attribute Lists

sound must contain one attribute, *src*.

define has two attributes, *word* and *phonemes*.

5.2.2 SSML Document Instance

The clearest way to describe an SSML document instance is by way of example. Figure 5–2 therefore shows an example document instance which conforms to the SSML DTD in figure 5–1. Each element defined in the DTD appears at least once.

I will go through the document in the order in which it appears on the page, describing each element as it appears.

```
<!doctype ssml system "SSML.dtd"
[]>
<ssml>
   <define word="buccleuch" phonemes="nil b uh . k l *oo">
   <phrase>
       I heard a <emph>noise</emph> on buccleuch
       place which sounded like this
   </phrase>
   <sound src="siren">
   <phrase>it was very loud
</ssml>
```

Figure 5–2: SSML Document

<ssml> ... </ssml>

This start-tag announces that what follows is marked up according to SSML. For the system as it stands at present, this is just a structural requirement, and serves no practical purpose, but in the future, it would be desirable to be able to contain SSML documents within other documents (an HTML document on the World Wide Web, for example). This would then be the tag which specifies which DTD is to be consulted for the portion of document which follows, until the end-tag. This and other possibilities are discussed in chapter 7.

<define ... >

As this element has been defined as empty, it consists only of a start-tag containing attribute values. The first of these, *word*, is followed by the spelling of a word whose pronunciation is being defined or redefined. The second, *phonemes*, specifies the desired pronunciation. This facility allows the user to specify a pronunciation for a particular word; this pronunciation will be used from that point on, unless the

same word is later redefined, at which point the new definition will overwrite the previous one. It may be that the word is already in the lexicon, but that the user would like it to be pronounced differently in this instance, or it may be that this is a new word which is being introduced by the user. The pronunciation must of course be specified using the conventions of the lexicon which is currently in use. Information on the conventions of the lexicon will be contained in a style sheet (7.4).

<phrase> ... (</phrase>)

The text is divided into phrases, and text may not occur outside a phrase. The first phrase which appears in figure 5–2 contains an emphasized word, and has an explicit end-tag. The second one does not specify which word is to be emphasized — this is also allowed by the DTD — and the end-tag is implicit. This is because it is always possible to work out where a phrase ends, as it will always be followed immediately by another phrase, a sound, a define or an SSML end-tag. When the document is parsed, the end-tag will appear in the output (see section 6.3).

<emph> ... </emph>

When the speech is synthesized, this word contained within an emph element will be emphasized. This holds only for this particular instance of any given word; if it is encountered again later in the text, it will not be given special treatment.

<sound ... >

This is the facility which allows the user to insert any sound contained in another file into the middle of an SSML document. Like define, sound is an empty element. Its only attribute, *src*, gives the name of the file from which a sound is to be taken. The sound will be played at the specified moment in between synthesized spoken phrases.

Chapter 6

From Markup to Synthesis

In this chapter, I describe the processes which much be applied to a document before it can be spoken by a synthesizer. These processes can be divided into two stages, one of which will be common to all systems, and the other of which will be specific to the particular synthesizer being used.

The first stage is the parsing of the document using an SGML parser (see section 4.5). The **nsgmls** application of James Clark's SP parser[1] which was used for this first stage is described in section 6.3. The second stage is the processing of the parsed output into a form which can be understood by the synthesizer. With this prototype version of SSML the CSTR speech synthesizer was used, and it is described in section 6.1.

The MTS program written for this project, which performs the processing of the parsed document, was implemented in C++, and is described in detail in section 6.4.

---

[1]This and other SGML parsers are freely available on the World Wide Web, see the SGML www pages [SG] for details. (www addresses valid as of 11 September 1995.)

6.1 The CSTR Synthesizer

The CSTR synthesizer consists of two modules, a linguistic component and a waveform generator. In reality, these modules are totally separate from one another, but for the purposes of this dissertation, they will be referred to as "the CSTR synthesizer". The output from the linguistic component is fed into the waveform generator, which then outputs the speech. The waveform generator is a diphone synthesizer, and this will be described first; then there will be a description of the linguistic module, which is implemented using a stream-based architecture.

6.1.1 Diphone Synthesis

The CSTR diphone synthesizer is part of the Osprey speech synthesis system and is described in [Tay92].

6.1.2 Stream-based Architecture

The linguistic component of the CSTR synthesizer uses a stream-based architecture, in contrast to many traditional TTS systems, where a linear architecture is used. In these systems, such as MITalk [AHK87], information is passed through one module at a time, each module being a set of algorithms which operates on well-defined input and output structures. The first stream-based architecture was the Delta[2] Rule Development System for Speech Synthesis from Text [Her90], and the current CSTR implementation, designed by Paul Taylor of CSTR, is an improvement on a similar architecture which forms the basis of the CHATR speech

[2]The name "Delta" was used because a geographical terms a delta is a location where several streams meet.

synthesis system [BT94]. The stream-based approach is based on a blackboard architecture, where all modules have access to the information at all times. An utterance object is created, which consists of a number of streams, each of which contains a list of items. Each stream will contain only items of a certain linguistic type. Each item in the stream contains all the information relevant to its particular linguistic type, and also a list of links to items in other streams. A diagram of an utterance object which has four streams is shown in figure 6–1.

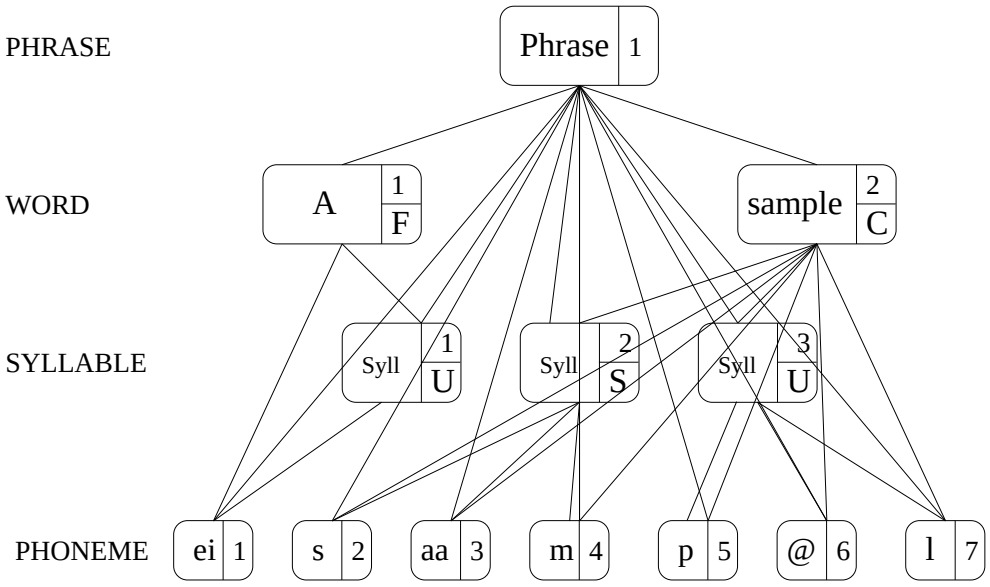

Figure 6–1: Stream Architecture for "A sample"

The phrase stream contains 1 item.

The word stream contains 2 items.

The syllable stream contains 3 items.

The phoneme stream contains 7 items.

In this example the utterance object contains only one phrase, but it could contain any number. Each stream-item has a unique identifier within its own stream; in

this case, the items are numbered starting from 1 in each stream, but any system could be used. The word and syllable stream-items carry extra information in this example. Each word is identified as a function (F) word, or a content (C) word. Each Syllable is identified as being stressed (S) or unstressed (U).

Phrase 1 is linked to all the other items in all the streams in this example, as it is the only phrase in its stream.

Word 1 ("A") is a function word, and is linked to Syllable 1 and Phoneme 1.

Word 2 ("sample") is a content word, and is linked to Syllables 2 and 3, and to Phonemes 2 3 4 5 6 and 7.

Syllable 1 is unstressed, and linked to Phoneme 1.

Syllable 2 is stressed, and linked to Phonemes 2 3 and 4.

Syllable 3 is unstressed, and linked to Phonemes 5 6 and 7.

This architecture is much more flexible than a traditional pipeline architecture because input and output structures do not have to be defined for each module. When a system is written using a stream-based architecture, no assumptions are made about the structure of the modules which will be included in the system, and it is therefore not necessary to make major changes to the system to allow the incorporation of new or significantly redesigned modules.

6.2 Emacs psgml mode

All of the preparation of SSML documents was carried out using psgml,[3] which is a major mode for the **emacs** text editor. This mode facilitates the creation and checking of SGML documents using pull-down menus, and includes a validating parser (see section 4.5.1).

6.3 Nsgmls

Nsgmls is an application of James Clark's **SP** parser. It both parses and validates a file, printing on the standard output error messages concerning any errors in the document and a version of the document in canonical form, with all end-tags supplied (see section 4.5). Since documents have usually been prepared using the emacs psgml mode, there should be no error messages. The output from the SSML document in figure 5–2 is shown in figure 6–2.

Once this stage has been reached, the output is ready to be sent to a program specifically designed for the synthesizer which is in use. The MTS program written for this MSc project to perform this task for the CSTR synthesizer is described in the next section.

[3]For details of how to obtain psgml, see the SGML www pages [SG] (www addresses valid as of 11 September 1995).

```
(SSML
AWORD CDATA buccleuch
APHONEMES CDATA b uh k l *oo
(DEFINE
)DEFINE
(PHRASE
-I heard a
(EMPH
-noise
)EMPH
- on buccleuch place which sounded like this
)PHRASE
ASRC CDATA siren
(SOUND
)SOUND
(PHRASE
-it was very loud
)PHRASE
)SSML
C
```

Figure 6–2: Parsed SSML Document

6.4 MTS

MTS (markup-to-speech) takes the output from **nsgmls** as shown in figure 6–2, and converts it into a form suitable for the linguistic module which attaches to the CSTR synthesizer, which was described in section 6.1.1.

There are several command line options:

-o<file> Specifies the output file. Two files are created with this name. The first, "<file>.utt", contains the full utterance structures which have been created by the mts program (see sections 6.1.2 and 6.4.1. The second, "<file>.wav", contains the processed data which can be fed straight into the synthesizer.

-nosynth Do not carry out synthesis (no ".wav" file will be created).

-sil Perform all the operations necessary for the synthesis, but do not output the sound.

6.4.1 The Streams

As described in 6.1.2, the CSTR synthesizer takes as input a number of streams linked to one another. The streams which need to be filled are:

- Phrase
- Word
- Syllable
- Phoneme
- Event

- Emphasis

Phrase, word, syllable and phoneme are fairly self-explanatory, and each item in each of these streams is linked to at least one item in each of the others. All links are bi-directional. The emphasis stream will always contain 3 items: two are empty and linked only to the phrase, while the other is linked to the word which is emphasized and the phrase. The event stream will also contain three items, two of which will be empty and one linked to the stressed syllable of the emphasized word.

6.4.2 Structure of the Program

The program works by processing each line of the output of the **nsgmls** parser in turn, and applying suitable actions.

Lexicon

First, the lexicon is read in, and stored in a key-value list, with the graphemic form of the word as key, and the phonemic and stress information as value. The lexicon which I have used is the CSTR TTS lexicon, and some typical entries look like this:

an *fn* a n

avocado *nil* ˜a . v @ . k *aa . d ou

is *fn* i z

unmistakable *nil* ˜uh n . m i . s t *ei . k @ . b @ l

The spelling of each word is in bold. The section in italics states whether the word being defined is a function (*fn*) or content (*nil*) word. The remainder of the line is a phonemic description of the word with stress and syllable information.

```
BEGIN
create empty streams
create empty lexicon and special lexicon

WHILE (there is still input to be processed)
  {
    IF (lexicon)
      read the specified lexicon into a key-value list
    IF (phoneme definition)
      add word and new definition to special lexicon
    IF (sound file to be played)
      play file
    IF (beginning of phrase)
      initialise item in phrase stream
    IF (text)
      process text, adding items to word, syllable, phoneme
      streams and linking as necessary
    IF (emphasized word)
      initialise item in emphasis and event streams, and link as
      necessary
    IF (end of phrase)
      send phrase to synthesizer
  }
END
```

Figure 6–3: Pseudo-code for the MTS Program

. marks a boundary between syllables.

* before a phoneme marks primary stress on that phoneme

˜ before a phoneme marks secondary stress on that phoneme

Special Lexicon

When the program is started, a special lexicon is created, which will contain entries for any words which are defined in the markup. It is possible to redefine the same word any number of times; an existing entry which contains the same key as a new one will simply be overwritten. This makes it possible to change the pronunciation of a word during a document, so that it can be spoken in several varying ways at different times. Entries to the special lexicon must be made in the same format as that used in the main lexicon.

Sound Files

A sound file which has been included in the document will simply be sent to another program which will play it immediately.

Emphasis

When a word is to be emphasized, an emphasis item and an event item are added, and both are linked to the phrase stream and to each other. An emphasis flag is then set to 1, so that when the following text is being processed, the emphasis and event items will be linked to the appropriate word and syllable items respectively. As soon as this has been done, the emphasis flag is reset to 0.

Processing the Text

Each section of text is processed in several stages.

Word To begin with, the words are separated and each one is dealt with in turn. The word is looked up first in the special lexicon, which contains the phoneme definitions specified in the current document, and if it is not found there, it is looked up in the main lexicon.

If it does not appear in either of the lexicons, it is added to a list of unknown words which is printed out after the synthesis has been completed, and attention passes to the next word.

If the word is found in one of the lexicons, a word item is created, and it is linked to the most recently created phrase item.

If the emphasis flag is currently set to 1, the word is also linked to the most recently created emphasis item. The value of the word is then retrieved from the lexicon. As described in section 6.4.2 this contains information on whether the word is a content word or a function word, and a phoneme definition. The content/function information is included in the word item when it is created.

The phoneme definition is processed to extract syllable information, and when this has been done, each syllable is again processed in turn.

Syllable A syllable item is created and linked to the appropriate word and phrase items, and a marker is also included which states whether the syllable carries the primary stress of the word. If the emphasis flag is at 1, *and* the syllable carries the primary stress of the word, the syllable is linked to the most recently created event item.

Phoneme An item is then added for each phoneme, and each one is linked to the appropriate syllable, word, and phrase items.

Automatic Emphasis

It is necessary for the synthesis that each phrase contain one emphasized word, even if one has not been specified in the markup. To ensure that this occurs, when an end of phrase marker is found, a check is made to see whether the phrase item in question has an emphasis item already linked to it. If it does, all is well, and the phrase can be sent to the synthesizer. If not, emphasis must be added in a suitable place. An emphasis item and an event item are added to the streams, and both are linked to the most recently created phrase and to each other. It is common practice to assign emphasis to the last content word in a phrase if no emphasis has been explicitly specified,[4] so the word stream is searched to find the appropriate item, and the emphasis item is linked to it. The event item is linked to the stressed syllable of this word.

Speaking the Text

When a phrase has come to an end and all the streams have been filled, the complete utterance structure is sent to the synthesizer. As a sound played from another file can only occur outside a phrase, this means that the whole document will be synthesized in the order in which it appears on the page.

When the text is spoken, certain rules are applied when phrase boundaries or emphasized words are encountered [Tay93].

For phrase boundaries:

- A pause is inserted at the end of each phrase,

[4]The question of which word this emphasis should be added to is by no means a simple one, but a detailed discussion will not be included here. The subject is treated in depth in [Mon91].

- The final syllable in the phrase is lengthened,
- The F_o contour is dropped to the baseline of the speaker's pitch range.

For emphasized words:

- The accented syllable of the word is lengthened,
- There is a rise in pitch from the beginning, towards the middle of the accented word, followed by a fall to the end of the syllable.

Chapter 7

Extensions and Conclusions

In this chapter, I present an enlarged SSML DTD, and a discussion of the additional elements contained in it. I then conclude with an evaluation of the project as a whole.

7.1 Extended SSML DTD

The elements in figure 7–1, an extended SSML DTD, can be divided into categories depending on whether or not they could be easily implemented using current synthesizer technology .

7.2 Plausible Additions

Volume. This element can be given attributes of *loud, medium* or *quiet*, and it is up to the individual writing the conversion program to decide exactly what these should translate to in terms of amplitude. This is analogous to the way that <h1>, <h2> etc. title markers function in HTML — it is up to each individual browser to decided on the appearance of each level of title.

```
<!element ssml - - ((sound | define | mood | language
                     | style | pitchrange | volume
                     | speed | speaker)*,
                    phrase,
                    (phrase | sound | define | mood
                     | language | style | pitchrange
                     | volume | speed | speaker)*)>

<!element phrase - o (#PCDATA | emph)*>
<!element emph - - (#PCDATA)>
<!element sound - o EMPTY>
<!element define - o EMPTY>

<!element mood  - o EMPTY>
<!element language - o EMPTY>
<!element style - o EMPTY>
<!element pitchrange - o EMPTY>
<!element volume - o EMPTY>
<!element speed - o EMPTY>
<!element speaker - o EMPTY>

<!attlist sound src CDATA #REQUIRED>
<!attlist define word CDATA #REQUIRED
           phonemes CDATA #REQUIRED>
<!attlist phrase type (statement | question ) statement>
<!attlist mood mood (happy | sad | sarcastic | angry
                     | neutral) neutral>
<!attlist language language (english | french | german)
                            english>
<!attlist style style (reading | conversation | oratorial)
                      conversation>
<!attlist pitchrange range (high | medium | low)
                           medium>
<!attlist volume volume (loud | medium | quiet) medium>
<!attlist speed speed (fast | medium | slow) medium>
<!attlist speaker speaker (fred | louisa | henry)
                          #REQUIRED>
```

Figure 7–1: An Extended SSML DTD

It would also be possible for the attribute to be a number on a scale from 0 to 1, which would later be translated into actual values for the synthesizer.

Pitchrange. This element can be given attributes of *high, medium* or *low.*

Speed. This element can be given attributes of *fast, medium* or *slow.*

Language. This element allows the natural language of synthesis to be altered in the middle of a document. Suitable lexicons will already have been defined in the style sheet (see 7.4 below). Each language will have one or more lexicons asoociated with it, and each lexicon description will include the phonetic alphabet and other conventions used in it.

Speaker This allows one text to be spoken using several different voices, which have been specified in the style sheet.

Question vs Statement This changes the prosody of a phrase depending on whether the phrase is a declarative or an interrogative one [Tay93].

7.3 Not yet, but ...

Mood It has not yet been established exactly how humans convey different emotions to one another. It is, of course, not done entirely with the tone of voice; facial expressions and body language have a major effect on how a speech utterance will be interpreted. However, it is possible to convey some emotions reliably in synthetic speech [MA93].

Style Some research has already been done into different speaking styles. In [AM94] there is a discussion of the differences in style between the speech used in an advertisement, in reading a novel, and in speaking conversationally. [EI92] discusses the difference in style between casual and careful speech.

Sentence Types SS systems already commonly include different prosody for at least two types of sentence, most commonly declarative and interrogative [Ste85]. There are of course many more useful distinctions to be made, for exmaple between different kinds of interrogative sentences, such as "yes/no" questions and "wh word" questions.

Figure 7–2 shows an example document marked up using the extended DTD.

```
<!doctype ssml system "extended.dtd" []>
<ssml>
  <speaker speaker="louisa">
  <mood mood="sarcastic">
  <phrase><emph>what</emph>a nice hat</phrase>
  <speaker speaker="fred">
  <pitchrange range="high">
  <style style="conversation">
  <mood mood="angry">
  <phrase>shut the door<emph>now</emph></phrase>
</ssml>
```

Figure 7–2: An Extended SSML document

7.4 Style sheets

SGML documents are designed to be device-independent, and therefore to contain no information on how the processing of a given element is to be done. So, for example, it would not be acceptable to define an element **volume** which allows you to specify freely how loud the sound is to be in decibels ; you could on the other hand give the element **volume** a choice of attribute values for example *loud, quiet*, and leave further processing to the particular implementation. Sometimes

it becomes desirable to be able to specify actual implementational details, and the way this is done in SGML is with *style sheets* and *link elements* [Bry88] [Gol90]. Style sheets are separate files which contain details of how elements are to be formatted, and they are linked to documents using SGML entity declarations. In this way, specific formatting instructions remain separate from the actual document, but in a standard form. Different style sheets could be prepared for different synthesizers, or if a document was to be used both for text printing and speech synthesis, radically different style sheets could be prepared which would be linked to the same SSML document. If several languages are to be available, the specification of which lexicon is to be used with each language will be included in the style sheet.

It is sometimes difficult to decide whether a given aspect belongs in a style sheet or actually in the document. The language in which a document is to be spoken must appear within the actual document, but the particular accent in which it is to be spoken belongs in the style sheet. The boundary between different accents within a language and actual dialects is a fuzzy one, so this will not always be an easy distinction to make.

7.5 Conclusion

The initial aim of this project was to design and implement a prototype system which would define a markup standard for synthetic speech, and to examine the issues of what sort of markers the system should have, and make recommendations for a future system. This has been accomplished. A prototype SSML was written, and succesfully used with the CSTR speech synthesizer. This establishes that the concept of a Speech Synthesis Markup Language is a viable one. SSML could now be extended to include many more possibilities, and in this thesis, various ways in which this could be done have been discussed.

Bibliography

[AHK87] J. Allen, S. Hunnicut, and D. Klatt. *From Text to Speech: the MITalk System.* Cambridge University Press, 1987.

[AM94] Masanobu Abe and Hideyuki Mizuno. A strategy for changing speaking styles in text-to-speech systems. In *Proc. ESCA Workshop on Speech Synthesis, New York, USA*, 1994.

[Bry88] Martin Bryan. *SGML : an author's guide to the Standard Generalized Markup Language.* Addison-Wesley, 1988.

[BT94] Alan W. Black and Paul A. Taylor. CHATR: A generic speech synthesis system. In *COLING '94, Kyoto, Japan*, 1994.

[CG86] R. Carlson and B. Granstrom. Linguistic processing in the kth multilingual text-to-speech system. In *International Conference on Speech and Signal Processing*, volume 4, pages 2403–2406, Tokyo, Japan, 1986. IEEE.

[CM89] F. Charpentier and E. Moulines. Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones. In *Proc. Eurospeech '89, Paris*, pages 13–19, 1989.

[EI92] Maxine Eskénazi and Amy Isard. Characterizing the change from casual to careful style in spontaneous speech. *Journal of the Acoustical Society of America*, 90(2):2363–4(A), 1992.

[Ent] Entropic Research Laboratory, Inc., World Wide Web. *TrueTalk(tm): Text-to-Speech Technology*. http://www.entropic.com/ (Sept '95).

[Fla84] James Flanagan. Voices of men and machines. In Geoff Bristow, editor, *Electronic Speech Synthesis*, pages 48 – 66. Granada, 1984.

[GMS94] C. Günther, C. Maienborn, and A. Schopp. A cognitive motivated approach to a concept-to-speech system. In *Proc. ESCA Workshop on Speech Synthesis, New York, U.S.A*, 1994.

[Gol90] Charles F. Goldfarb. *The SGML Handbook*. Oxford University Press, 1990.

[Her90] Susan R. Hertz. The delta programming language: an integrated approach to non-linear phonology, phonetics and speech synthesis. In John Kingston and Mary E. Beckman, editors, *Papers in Laboratory Phonology 1*. Cambridge University Press, 1990.

[IM86] Stephen D. Isard and D. A. Miller. Diphone synthesis techniques. In *IEE Conference Publication no 258*, pages 77–82, 1986.

[Kla87] Denis Klatt. Review of text-to-speech conversion for English. *Journal of the Acoustical Society of America*, 82(3):737–793, 1987.

[MA93] I. R. Murray and J. L. Arnott. Towards the simulation of emotion in synthetic speech: a review of the literature on human vocal emotion. *Journal of the Acoustical Society of America*, pages 1097–1108, 1993.

[MB94] C.M. Sperberg McQueen and Lou Burnard, editors. *Guidelines for Electronic Text Encoding and Interchange (TEI P3)*, chapter 2. ACH/ACL/ALLC, Chicago, April 1994.

[Mon91] Alex Monaghan. *Intonation in a Text to Speech Conversion System*. PhD thesis, University of Edinburgh, 1991.

[Ram91] T.V. Raman. *Audio System for Technical Readings.* PhD thesis, Cornell University, 1991.

[SG] *The SGML Web Page.* World Wide Web. Maintained by Robin Cover. http://www.sil.org/sgml/sgml.html (Sept 95).

[Ste85] Michel Stella. Speech synthesis. In Frank Fallside and William Woods, editors, *Computer Speech Processing.* Prentice Hall International, 1985.

[Tay92] Paul A. Taylor. The Osprey speech synthesis system. Technical report, CSTR, 1992.

[Tay93] Paul A. Taylor. Synthesizing intonation using the RFC model. In *Proc. ESCA Workshop on Prosody, Lund, Sweden*, 1993.

[Tra95] Christof Traber. *SVOX: The Implementation of a Text-to-Speech System for German.* PhD thesis, Swiss Federal Institute of Technology, 1995. Published by vdf Hochschulverlag AG an der ETH Zürich.

[vH90] Eric van Herwijnen. *Practical SGML.* Kluwer Academic Publishers, 1990.

Appendix A

SGML Bibliography

- The SGML Web page [SG]

  This contains links to many pages of information.

- The TEI Guidelines for Electronic Text Encoding and Interchange [MB94]

  A very clear introduction to how to compose an SGML document.

- The SGML Handbook [Gol90]

  A reference text which includes the entire text of the ISO standard, divided up and annotated.

- Practical SGML [vH90]

- SGML: an author's guide to the Standard Generalized Markup Language [Bry88]

Appendix B

Glossary of Abbreviations

These are the abbreviations commonly used in this thesis:

SSML Speech Synthesis Markup Language.

SGML Standard Generalized Markup Language.

TTS Text-to-Speech.

SS Speech Synthesis.

DTD Document Type Definition.

HTML Hypertext Markup Language.

CSTR The Centre for Speech Technology Research, Edinburgh University.

MTS Markup-to-Speech