SCIENTIFIC PAPERS
OF THE UNIVERSITY OF PARDUBICE
Series B
The Jan Perner Transport Faculty
12 (2006)

EXTENDING ANNOTATIONAL SSML INTO STRUCTURAL CONTENT FOR SPEECH SYNTHESIZER

Martin KLIMO, Igor MIHÁLIK

Department of information networks, Faculty of management sciences and informatics,
University of Žilina

1. Introduction

The most common type of input to a speech synthesizer is a 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 analyze 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 emphasized, 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 emphasized, phrase breaks can be placed and instructions can be given which indicate how words should be pronounced.

There exists a standardized way of annotating text and it is called Speech Synthesis Markup Language (SSML)[1]. It is designed to provide a rich, XML-based markup language for assisting the generation of synthesis speech in Web and other applications. The essential role of the markup language is to provide authors of synthesizable content a standard way to control aspects of speech such as pronunciation, volume, pitch, rate, etc. across different synthesis-capable platforms.

2. SSML Document Processing

A text document provided as input to the synthesis processor may be produced automatically, by human authoring, or through a combination of these forms. SSML defines a form of the document. The following are the six major processing steps undertaken by a synthesis processor to convert marked-up text input into automatically generated voice output. The markup language is designed to be sufficiently rich so as to allow control over each of the steps described below so that the document author can control the final voice output: *XML parse, structure analyses, text normalization, text to phone conversion, prosody analyses and waveform production*.

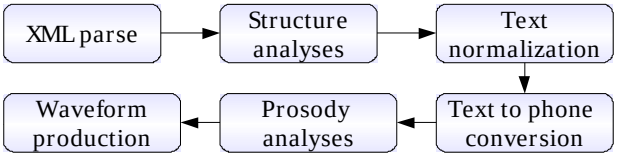

Fig. 1 *Steps needed during the process of speech synthesis*

Text to speech system presented in this paper covers all previously mentioned steps. The system is able to process SSML input directly and following sections will describe more in detail interesting conceptual and implementation aspects of the synthesizer. Also the extension to SSML language will be presented to give more fine grained annotations of input text so that the waveform output sounds even more naturally.

3. SSML Processing

As mentioned before SSML is the primary relevant input to synthesizer. The SSML does not give put any constraints on how detailed the annotating should be. One can write SSML like this one:

```
<speak>
  This is ordinary text. Whatever can be written here and it is a valid and
well
  formed SSML document.
</speak>
```

To better understand why this is possible one has to take SSML mainly as an **annotational** rather than structural tool. Its purpose is not to define tight structure of input text, but rather allow providing additional informational data (*metadata*) where it makes

sense to do so. If one wants to specify that the content between <speak> elements contains two paragraphs the input looks like this:

```
<speak>
  <p>This is ordinary text.</p>
  <p>Whatever can be written here and it is a valid and well formed SSML
document.</p>
</speak>
```

User could go deeper and annotate entire text so that the waveform output corresponds more to the user's expectations. This is an even more detailed example of annotated text with elements defined in SSML:

```
<speak xml:lang=”en-US”>
  <lexicon uri="http://www.example.com/lexicon.file"/>
  <voice gender="female" variant="2">
    <p><s>This is ordinary text.</s></p>
    <p><s>Whatever can be written here and it is a valid and well formed SSML
        document.</s></p>
  </voice>
 </speak>
```

As shown SSML gives possibility to set a lexicon that will be used for grapheme to phonetic transcription, to select a voice used during the speech rendering also sentence boundaries. The annotations that SSML provides can be divided into three groups depending on aspect they cover:

- Document structure, text processing and pronunciation (“speak”, “language”, “lexicon”, “phoneme”, “say-as”, “p”, “s”…)
- Prosody and style (“voice”, “emphasis”, “break”, “prosody”)
- Other elements (“audio”, “mark”, “desc”)

4. From annotational SSML to structural content

Any implementation of speech synthesizer (and any software product generally) requires analytical and architectural phase. During these phases internal architecture is defined, the modularity and interoperability between various modules is specified (communication interfaces). And when taking all of these aspects into account we discovered that SSML is a perfect basis for the definition of internal structure. Any XML [3] document (including SSML) is a linearization of tree structure. Each XML element can serve as a parent element or can be nested inside another XML element. SSML input shown in previous text can be depicted as tree:

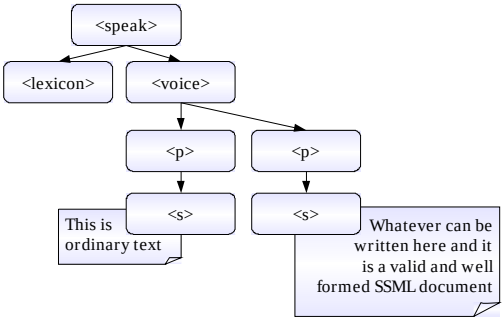

Fig. 2 *Tree like structure of SSML input*

Having this basis one needs to define a complete tree with all the required elements and their attributes. To achieve this we introduced several new elements and attributes and named this definition as SSML+:

- Compound element for annotating compounds in input text
- Type attribute to set a sentence type (declarative, interrogative, imperative)
- n-phone element to denote phonetic transcription (not as annotation above text, but rather element inside syllable element)
- syllable element to denote syllable inside a word

In order to achieve complete SSML+, structure "creational" modules were defined and implemented. In other words if input SSML text does not contain all necessary annotations there must be a module that is able to set annotations automatically. Rule-based engines (RBE [5]) represent a way how this task can be accomplished. We've defined a set of atomically and independently executable modules. Each module M is determined by conditions:

- *preconditions* – a set of conditions that have to be met before a module code is executed, denoted as
- *postconditions* - a set of conditions that must be always true just after the execution of module code, denoted as

These modules are executed one by one in undefined order depending on the result of module preconditions. This method of execution in which the order of module invocations is not explicitly defined is called *implicit invocation* [7]. The preconditions and postconditions determine implicitly the order in which modules are executed.

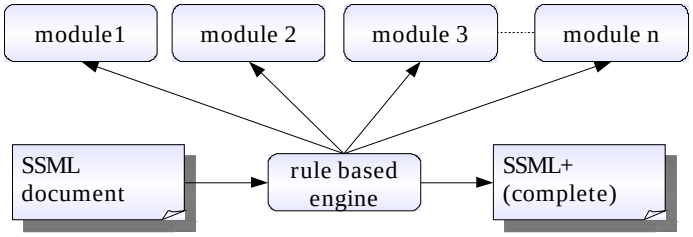

Fig. 3 *Precondition based execution of modules*

Let's have a set of modules:

$$M = \{M_1, M_2, M_3, ...M_n\} \tag{1}$$

With preconditions and postconditions:

$$P = \{P(M_1), P(M_2), P(M_3), ...P(M_n)\} \tag{2}$$

$$R = \{R(M_1), R(M_2), R(M_3), ...R(M_n)\} \tag{3}$$

To make sure that output of set of modules **M** produces complete SSML+ the postconditions of modules **M** must quarantine that output from modules leads to complete SSML+.

The approach used to determine these postconditions is based on backward chaining algorithm from artificial intelligence theory [7]. In terms of preconditions we have to guarantee that for each module there is a subset **M'** of modules **M** such that the unification of postconditions triggers modules precondition. And of course there must be one module in an implicit chain that is triggered by any valid SSML document.

Such modularization brings better understanding of system. The simplicity gives researchers more time spending on solving the problem each module should handle without knowing internals of other modules.

Preconditions and postconditions in fact define a contract of module interfaces and this approach is known as design by contract [10].

5. Formal language

To be able to formally write and reason about preconditions and postconditions a formal language was defined. Using this language one can create propositional functions with true predicates for elements of some subset of all elements in a tree. Then preconditions and postconditions are fulfilled.

The basic form of expression is . Where x represents element of tree, $\varphi(x)$ is propositional function with true or false result when applied on element x. Using basic form of propositional function and logical functions one can create complex formulas. One often needs quantifiers for some types of propositional functions:

- $\forall x \in A$ represents propositional function $\bigwedge_{x \in A}$ and we will write: For all holds or for each holds .
- $\exists x \in A$ represents propositional function $\bigvee_{x \in A}$ and we will write: There exists that holds
- $\exists! x \in A$
  There exists exactly one that holds

6. Modules Overview

Last descriptive section of this paper is focused on short enumeration of modules used in speech synthesizer. The speech synthesizer uses concatenative approach[8,9] and some of the modules are used cause of the concatenative method:

- Validation of SSML and structure creation (speak, ssml, audio)
- Structural analyses (lang, voice, paragraph, compound, sentence, sentence type, word, diphone, text norm, text, syllable, say as, sub, desc, mark, phoneme)
- Grapheme to phoneme conversion (lexicon, ph, word join)
- Prosody analyses (prosody, contours, range, pitch, duration, rate, volume, emphasis, voice [age/gender], break)
- Waveform production (diphone extract, diphone selection, diphone merge, contour application, prosody application)

7. Conclusion

We presented the principle and internal design ideas of speech synthesizer developed at Department of Information Networks. Although the implementation is focused on Slovak language the principles are applicable to any speech synthesizer with the aim of producing high quality speech output. Having this skeleton structure of data and modules we aim to focus deeper into research and implementation of each module allowing us to tackle specialties of each separately.

Lektoroval: prof. Ing. Karel Šotek, CSc.
Předloženo: 28.2.2007

References

1. Speech Synthesis Markup Language (SSML) Version 1.0, http://www.w3.org/TR/speech-synthesis
2. The Festival Speech Synthesis System, http://www.cstr.ed.ac.uk/projects/festival/
3. Extensible Markup Language, http://www.w3.org/XML/
4. Associating Style Sheets with XML documents, http://www.w3.org/TR/xml-stylesheet/

5. Rule Base Engine, http://en.wikipedia.org/wiki/Rules_engine
6. An Introduction to Implicit Invocation Architectures, Benjamin Edwards. http://www.mach-ii.com/downloads/docs/Intro%20to%20Implicit%20Invocation.pdf
7. AI Application Programming, M. Tim Jones, 2003 Charles River Media, ISBN 1584502789
8. Improvements in Speech Synthesis: Cost 258: The Naturalness of Synthetic Speech, Eric Keller, 2002 John Wiley and Sons, ISBN 0471499854
9. Text, Speech and Dialogue 2004: 7th International Conference, TSD 2004, Text to Speech for Slovak Language, pp. 291-298, ISBN 978-3-540-23049-6 Caky, Klimo, Mihalik, Mladsik, 2004 Springer, ISBN 3540230491
10. Applying “Design by Contract“, Bertrand Meyer, IEEE Computer, October 1992, Pages 40-51

Resumé

ROZŠÍRENIE ANOTAČNÉHO JAZYKA SSML NA ŠTRUKTURÁLNY OBSAH PRE REČOVÝ SYNTETIZÁTOR

Martin Klimo, Igor Mihalik

Článok popisuje spôsob rozšírenia štandardu jazyk SSML, ktorý je primárne určený pre anotáciu vstupného textu na plne štrukturovaný obsah interne použitý v systéme syntézy reči. Vysvetľuje použitý prístup pre spracovanie takého SSML vstupu s použitím technológie "rule-based enginu" a implicitnej invokácie. Pre reasoning v implicitnej invokácii bol navrhnutý formálny jazyk a tento jazyk je taktiež v krátkosti v článku popísaný.

Resume

EXTENDING ANNOTATIONAL SSML INTO STRUCTURAL CONTENT FOR SPEECH SYNTHESIZER

Martin Klimo, Igor Mihalik

This paper describes a way of extending standard SSML language that is primary used for annotating input text into fully structural content used internally for speech synthesis. It explains approach used for processing such SSML input using rule based engines technology and implicit invocation. For reasoning in implicit invocation a formal language had to be created. This formalism is also shortly described in this paper.

Zusammenfassung

DIE VERBREITUNG DES SSML-STANDARD, DER FÜR VOLL STRUKTURIERTEN INHALT IM SYSTEM DER SPRACHSYNTHESE VERWENDET WIRD

Martin Klimo, Igor Mihalik

Der Artikel beschreibt die Verbreitung des SSML-Standard, der für voll strukturierten Inhalt im System der Sprachsynthese verwendet wird. Der SSML Standard ist primär für Vermerke der Input-Texte bestimmt. Im Artikel wird die Art der Verarbeitung solchen SSML-Input unter Verwendung der rule based engine und implicit invocation erklärt. Für das Reasoning in implicit invocation wurde eine formale Sprache vorgeschlagen, die auch im Artikel kurz erklärt wird..