In Proceedings of the 4th ISCA Tutorial and Workshop on Speech Synthesis (SSW4), Perthshire, Scotland, September 2001, pp. 167-172

The DEMOSTHeNES Speech Composer

Gerasimos Xydas and Georgios Kouroupetroglou

University of Athens, Department of Informatics and Telecommunications
Division of Communication and Signal Processing
Panepistimiopolis, Ilisia, GR-15784 Athens, Greece
{gxydas, koupe}@di.uoa.gr

Abstract

In this paper we present the design and development of a modular and scalable speech composer named DEMOSTHeNES. It has been designed for converting plain or formatted text (e.g. HMTL) to a combination of speech and audio signals. DEMOSTHeNES' architecture constitutes an extension to current Text-to-Speech systems' structure that enables an open set of module-defined functions to interact with the under processing text at any stage of the text-to-speech conversion. Details on its implementation are given here. Furthermore, we present some techniques for text handling and prosody generation using DEMOSTHeNES.

1. Introduction

A number of modular Text-to-Speech (TtS) systems have been developed during the last years, like CHATR [1], FESTIVAL [2] and EULER [3]. The two major issues for such architectures are how to accommodate the plethora of different linguistic representations and how to make an efficient usage of the information that these representations carry. Both issues have been well dealt by the FESTIVAL system via the introduction of the Heterogeneous Relation Graph (HRG) [4].

DEMOSTHeNES Speech Composer [5] has been carefully designed in order to be a scalable system, flexible to modifications. The core of DEMOSTHeNES is based on the HRG, which was first implemented in FESTIVAL as its basic UTTERANCE structure. However, the architecture of DEMOSTHeNES differs, in order to enable the implementation of the e-TSA Composer presented in [6] and [7], but also to allow a more functional communication between the various modules of the system.

An example of the importance of that issue is this: Some TtS applications support the insertion of a small set of "tags" within a document, in order to change their auditory behavior. For example, the tag <slow> in the text

```
His name is <slow>John</slow>.
```

allows in some applications the user to slow down the speaking rate. However, these "tags" are very specific and usually the systems that use them have a monolithic architecture and thus they are easy to be interpreted. On the other hand, modular architectures need a more flexible mechanism for embedding such "tags" in a text. Moreover, these "tags" cannot be pre-defined, but should be defined according to the available functionality of the system. Our approach does not target to the provision of an open set of such "tags" to the user, like mark-up languages like the VoiceXML[8] do. According to DEMOSTHeNES specifications, the system has to generate them by the source text. Thus, we call these "tags" *embedded instructions*.

Another issue that raised the need for an extension to current architectures came from the fact that modern TtS systems ([2] and [3]) manipulate the information not in a raw but in more complex and linked structures (e.g. metrical trees, MLDS etc). Thus, there should be a mechanism for synchronizing the ordinary information that is being analyzed with any embedded instruction.

The rest of this paper is organized as follows: in paragraph 2 we present the architecture of DEMOSTHeNES. In paragraph 3 and 4 we present specific implementation of text handling and prosody generation.

2. Architecture

DEMOSTHeNES is a modular and open system. Its functionality is defined by customized plug-ins, the *modules*. Each module can implement an arbitrary number of linguistic, phonological, acoustical etc functions. However, they need a mean for communicate with each other and exchange functionality. This is being done by the *kernel* and its *components* that store the shared data that modules exchange.

Thus, there are three basic elements (classes) in the architecture of DEMOSTHeNES (prefix 'V' in the following terms stands for 'Vocal'):

- VSERVER, which is a communication channel for the rest elements,
- VCOM (component), which provides essential structures and services concerning linguistic, phonological, acoustical etc procedures and
- VMOD (module), which inherits from VCOM and manipulates the structures of VCOMs, implements extra functionality and defines the behavior of the system, as a linked element (plug-ins).

We will present them in the next paragraphs in detail. The basic diagram of this architecture is given in Figure 1. This scheme is very scalable in terms of functionality and performance, as it can be downscaled to meet specific hardware specifications and also allows modules to be inserted and removed, modifying the capabilities of the system.

2.1. Vocal Server (VSERVER)

VSERVER actual implements a Directory Service where all the available functionality offered by VCOMs and VMODs is stored. The Directory Service reserves a namespace for each VCOM (and VMOD), where it keeps information (signatures) about the name and the address of the services they offer as

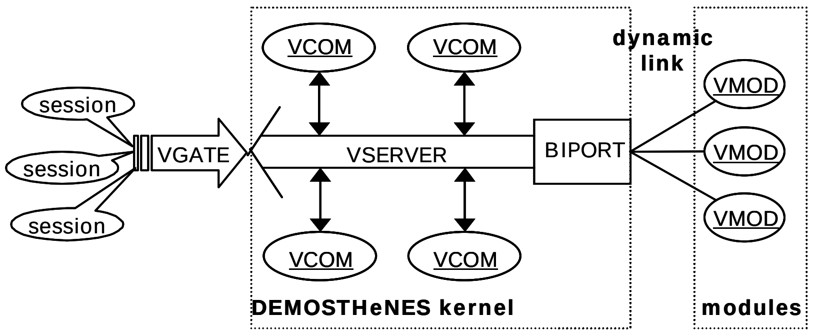

Fig. 1: The architecture of DEMOSTHeNES Speech Composer.

well as their attributes. For example, a VMOD can look up for the VCOM that implements the Heterogeneous Relation Graph (HRG). Upon getting its signature from the Directory Service, it can ask for the service that sequentially parses the nodes of a particular linguistic level (e.g. the "syllable" level) in order to process them. VSERVER is the linchpin of all the elements in this architecture. This is a difference between the architecture of DEMOSTHeNES and FESTIVAL: in the latter, modules normally communicate by passing references to an UTTERANCE object. In DEMOSTHeNES, modules ask for specific services.

2.2. Vocal Component (VCOM)

Specific care has been set for enhancing the flexibility of the system. Thus, it is possible to implement all the functionality of a TtS system using only VMODs. However, there are essential data structures and services in the procedure of converting text to speech that are useful or shared across different VMODs. This global and shared character of particular data and methods has been wrapped in the VCOM elements of DEMOSTHeNES. A sample list of implemented components follows: HRG, Decision Trees, Pitch, Time, CART, User Profile, Audio, Security and Interface. VCOMs are currently hard coded elements and if modifications are needed, major compilation should taken place.

2.3. Vocal Module (VMOD)

Further manipulation of the data stored in VCOMs is being done by the modules (VMODs). The VMOD class inherits from VCOM. There are two differences: (a) VCOMs have a global scope, while VMODs have not and (b) VMOD offer a very light interface (described bellow) while VCOM's is more detailed.

VMODs are placed in a logical sequence (chain), which defines the actual process of the text-to-speech conversion. They do not communicate with each other, apart from the option of using the Exported Methods of other VMODs (see §2.4.1). This is a simple chain:

```
vserver->addModule(vmod_infeeder);
vserver->addModule(vmod_text2sound);
vserver->addModule(vmod_infilter);
vserver->addModule(vmod_syllablation);
vserver->addModule(vmod_wordation);
vserver->addModule(vmod_phrasation);
vserver->addModule(vmod_dur_cart);
vserver->addModule(vmod_pitch_syllable);
vserver->addModule(vmod_pros_breath);
vserver->addModule(vmod_mbrola);
```

where `vserver` is a pointer to the unique instance of the VSERVER class, and any `vmod_xxxxxx` is an istance of a corresponding VMOD class. The names of the latter roughly present the functionality of each module ("dur" stands for duration, "pros" for prosody). This list also indicates the sequence of execution of each VMOD.

2.3.1. *Interface*

The interface of VMODs is very simple and consists of only three methods:

1. `initialize`(): all the significant initialization is taking place here. Called automatically after a successful registration.
2. `apply`(): the core of the module. When the chain is executed, the `apply`() method of each registered VMOD is called.
3. `finalize`(): called when the chain class is destroyed.

In contrast with the above, VCOMs offer more complicated and detailed interfaces to the system, in order for VMODs to take full advantage of them.

2.3.2. *Text annotation and reports*

Producing annotated text from a TtS system is very useful for corpus creation, as well as for creating training data. Trained models are widely used in TtS systems, especially for prosody generation and POS tagging. In DEMOSTHeNES, a source text can be exported as annotated, enriched by features and functions supported by the system. For example, giving a text we can create a document containing the syllables of the text, the next and the previous one for each syllable, their distance from the phrase boundaries, stress indication etc. These descriptions can be used for the creation of CART trees, for example for duration prediction when they are accompanied with the corresponding recordings and labels. CART trees are supported in DEMOSTHeNES, using a port for the WAGON tool (provided by Edinburgh Speech Tools [9]) output.

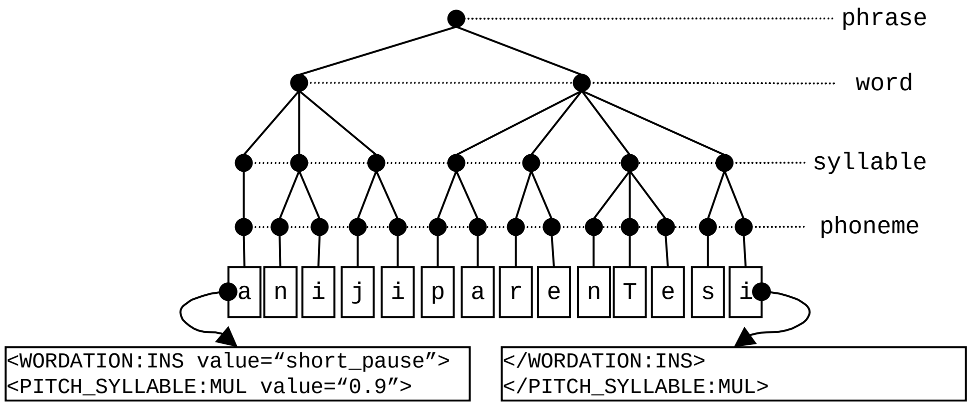
phrase
word
syllable
phoneme
a n i j i p a r e n T e s i
<WORDATION:INS value="short_pause">
<PITCH_SYLLABLE:MUL value="0.9">
</WORDATION:INS>
</PITCH_SYLLABLE:MUL>

Fig. 2 A sample HRG and the instruction-channel. "aniji parenTesi" stands for "open bracket" in Greek.

Furthermore, any VCOM (and thus any VMOD) is capable of creating a report of its state (for example in XML format). This is quite useful for debugging the system. Also, as HRG are supported, another interesting option is the export of UTTERANCE structures for exchanging information with the FESTIVAL system (e.g. to further use of the FRINGE tool [9] for data representation). Still, this is an ongoing procedure.

2.3.3. *Self benchmarking*

Text-to-speech conversion usually has soft and sometimes hard real-time requirements. This is more tangible in dialogue systems: the conversion should be feaster than the speech playback in order to avoid annoying pauses. Furthermore, devices with limited hardware (e.g. embedded devices) usually can not accommodate the requirements set by TtS systems.

In order to estimate the requirements of a TtS configuration using DEMOSTHeNES, the VMOD class offers a self-benchmarking procedure. Special variables indicate its CPU usage and the memory footprint during run-time. This is particularly useful for performance enhancements but also supports the scalability of the system: when the text-to-speech conversion fails to meet the real-time requirements (e.g. when process time is bigger than playback time => long pauses), highly consuming CPU modules can be automatically dropped.

2.4. Instruction Channel

The basic feature of this architecture is the introduction of an instruction channel that is synchronized to the ordinary flow of text information. Thus, in DEMOSTHeNES, the processing procedure flows in two synchronized channels: the first one (called "data-channel") carries all the linguistic, prosodic, phonological, acoustical, etc data in an HRG hierarchy, while the second one (called "instruction-channel") carries specific instructions for the VMODs. These instructions are linked to the leaves of a graph that represents the basic linguistic levels (called *level-graph*), as can be seen in the example of Figure 2. This way, when particular patterns are identified in the source text, we are able to program the behavior of the system using rules or scripts.

2.4.1. *Exported methods*

VMODs can optionally define a library of methods that are recorded to the Directory Service of VSERVER and can be called during run-time as embedded instructions to the source text. When a VMOD is added to the chain (as shown in §2.3), a registration phase is taking place, where it passes to the Directory Service a list of these exported methods. A VMOD can embed instructions to the source and interpret instructions that refer to it.

Embedded instructions are being interpreted by the corresponding VMODs, when their time for execution arrives. VSERVER notifies VMODs that instructions have been set, which in turn locate and execute the instructions while parsing the level-graph.

2.4.2. *Example*

We will present in detail the example of reading out a left bracket in a text, by decreasing the pitch and inserting a pause. In order to achieve this we use a service from the Finite State Transducer VCOM (presented in §3), which converts the source string

```
"("
```

to the:

```
"<WORDATION:INS
value="short_pause"></WORDATION:INS>
<PITCH_SYLLABLE:MUL value="0.9">open
bracket</PITCH_SYLLABLE:MUL>"
```

The above format is very similar to the format of XSLT templates [10]

- *WORDATION* is the namespace of the VMOD that creates the word level in the level-graph and set the features of each word,
- *INS* is an exported method of the *WORDATION* VMOD that inserts nodes to the particular level and
- *value="short_pause"* is the attribute of the above method

In a similar way, *MUL* is a method of the VMOD *PITCH_SYLLABLE* that multiplies the pitch values linked to the nodes of the syllable level by the value 0.9. This will be the behavior of the system everytime a left bracket is offered in the source text. The result string is much like an XML document and this is the base of the architecture of the e-TSA Composer ([6] and [7]).

Figure 2 presents the interaction of the level-graph with the instruction-channel. When the *WORDATION* module is executed, the Directory Service will call the *INS* method, when the corresponding phrase is being processed.

2.5. CART VCOM

In order to support trained models and take advantage of available tools, we have developed a bridge between DEMOSTHeNES and the WAGON, as already shown in §2.3.2. From the one side DEMOSTHeNES annotates recorded text, passes this data to WAGON and the produced CART tree is being translated to native code for DEMOSTHeNES. This procedure is being supported from the CART VCOM. CART trees are used for prosody generation, letter-to-sound rules, POS tagging etc. For example, the Greek configuration of DEMOSTHeNES supports the English language, using a letter-to-sound CART tree created from the CMU lexicon.

3. Text Handling

The text handling stage targets to an efficient representation of the source text to the rest stages of the Composer and finally to the speech output. The issues that are important here are to correctly identify particular string patterns in the given text and to successfully vocalize them. String patters comprise:

1. Numerics (numbers, dates, hours, telephone numbers, special formats like IP addresses)
2. Acronyms and other abbreviations
3. Marks (e.g. brackets, quotations etc)
4. Special characters (e.g. currencies, percentages, etc)
5. Language detection (e.g. English words, Italian words etc)

Text Handling in DEMOSTHeNES consists of a set of methods implemented in several VCOMs and VMODs that can identify, translate and embed instructions for further augmented auditory representation of the above patterns. In order to support this procedure, DEMOSTHeNES offers a Finite State Transducer engine implemented in a VCOM, for translating a source pattern to a target one under specific conditions using regular expressions. The general form of rules is:

$[X_1][c][X_2]\text{->}[str]:N$

which states that when character [c] is found in the context of expression [X_1] and [X_2], then it is converted to string [str]. N states the number of characters that are affected by this rule (including [c]). The format of [X_n] is as follows:

X_n:=″CHARACTER_SET″{REPETITION}, X_n

3.1. Example 1: Billions

```
[]["2"]["0123456789"{6}] -> ["two
billions"]:1
```

The above rule states that when we have the character “2” followed by any number (i.e. 1, 2, 3, 4, 5, 6, 7, 8, 9, 0) for 6 times (i.e. followed by 6 digits), then “2” is converted to “two billions” and this conversion affects 1 character (i.e. the character “2”).

3.2. Example 2: Acronyms

```
[" "{1}]["H"]["."{1}, "R"{1}, "."{1},
"G"{1}, "."{1}, "\0\n.,!;"{1}] ->
["Heterogeneous Relation Graph"]:6
```

The above rule states that when character “H” is after one space and is followed be one full stop, one “R”, another full stop, one “G”, one more full stop and any one of the marks “\0\n.,!;”, then it is translated to “Heterogeneous Relation Graph” and this rule affects 6 characters (i.e. “H.R.G.”).

4. Prosody

Prosody is probably the most important issue when generating speech from text. Even humans, with perfect pronunciation of foreign words, fail to sound like foreigners because they do not reproduce correctly the prosody of the foreign language, but they usually try to wrap foreign words in the prosody of their mother tongue. There are two major approaches to prosody generation: (a) rule based models and (b) stochastic models. In DEMOSTHeNES both models can be implemented among with any other model as VMODs. We will present an example of a rule-based implementation in §4.2, as we already showed how we deal with the training approach in §2.5.

DEMOSTHeNES provides a generic support for defining and handling prosody in terms of TIME, PITCH and AMPLITUDE (but any other prosodic feature can be implemented). These are offered as VCOMs among with mechanisms to be linked to the HRG.

4.1. Time and Pitch VCOMs

Time and Pitch is of particular interest in prosody generation, thus they are supported in VCOM level. Time is a linked coefficient to the leaves of HRG and several methods are offered by its VCOM regarding compression, expansion etc. Pitch is also a coefficient linked to the leaves of the HRG, however, Pitch offers a series of real values for each link. The number of the value frames depends on the Time (duration) associated with the specific leaf.

In DEMOSTHeNES Time and Pitch have a more close relation, as a linked mode has been designed. According to this mode, the duration of a syllable, for example, and its pitch curve can have an initial relation. If during processing, the pitch produces a big raise, then the duration for this segment will be expanded accordingly following a exponential relation. This way, we drive the duration of segments from the pitch movements.

4.2. Rule-based approach

We will show a rule-based approach for prosody generation that can have multilingual application. This has been implemented as a collection of VMODs. It is based on grouping words of sentences into smaller, more closely related groups, called *phrases*. The key for a successfully rendered prosody here can be summarized to the next sentence: “the wider the range of phrasal categories, the richer the produced prosody”. Which means, that we can achieve better prosody if we manage to smash a sentence to small meaningful phrases. Thus, the first step to this approach is to identify as much phrases as possible in a sentence. One or more VMODs can contribute to this purpose. Here is a list of candidate phrasal categories:

- Level A: Phrases seperated by marks (full stops, commas, exclamations, brackets, quotations etc). These are very easy to be identified and classified.
- Level B: Grammatical phrases, i.e. parts of the text starting from conjunctive words, pronouns etc and ending to another point of Level A or B. This words indicates secondary sentences and a POS tagger or a lexicon (as they can be quite few) is needed to identify them.
- Level C: Relative words. Usually some words have a very strong relation, like an article and a noun, an adjective and a noun, a verb and an adverb etc. These can form extra groups and require a POS tagger, in order to be identified.

Of course, someone can define an arbitrary number of levels, while the definitions are not mandatory. The next step is to assign intonation events to these categories according to the ToBI, the Tilt or any other intonation models and then interpret these marks in order to render the intonation curve. We add one more approach here, by assigning a Prosodic Template to each phrasal category. A Prosodic Template represents pitch and time (and amplitude if applicable) relations for the specific category. It can represent actual values, relative values, ToBI, Tilt labels, or any other annotation, as long as the annotation is capable of producing a sequence of time and pitch pairs under specific conditions.

We will present an example of a Prosodic Template for phrases that ends in a comma. Figure 3 shows an abstract pitch curve for the specific template:

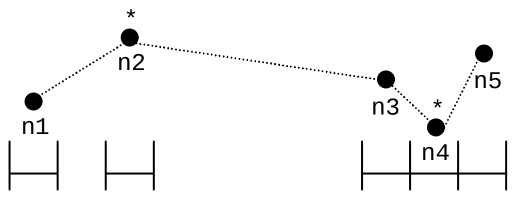

Fig. 3: An abstract pitch curve for "comma" phrases.

where `nx` are nodes in the "syllable" level and stars (*) indicate stress. It assumes that pitch transition happens per syllabic node. This abstract curve states that:

1. The curve starts from an arbitrary point, in the first node of the "syllable" level, n1.
2. Pitch is raised until the first stressed syllable in the phrase, n2.
3. In the last stressed syllable of the phrase, n4, the pitch is falling instead of raising, just before the comma,
4. and it is raised until reaching the comma, after n5.

This scenario has been made by observing intonation curves for the Greek and English language. Now, we will constract the template according to this abstract curve. We first locate the five nodes of interest:

```
NODE n1=head(level("syllable"));
NODE n2=n1.forward.until("stressed");
NODE n5=tail(level("syllable"));
NODE n4=n5.backward.until("stressed");
NODE n3=n4.prev();
```

We assume no special cases here (like n5==n4 etc). Next, we define a reference pair (pitch, time) of values for the above nodes. Pitch element represents a pitch reference, while time element a time reference throughout the whole node of the level of interest ("syllable").

```
n1.pt_reference(1.0, 1.0);
n2.pt_reference(1.2, 1.1);
n3.pt_reference(1.0, 1.0);
n4.pt_reference(0.8, 1.0);
n5.pt_reference(1.0, 1.2);
```

The above script roughly says that the first stressed syllable in the phrase (n2) should take a 20% increment in the pitch, while the last stressed syllable in the phrase (n4) should take a 20% decrement in the pitch (instead an increment). Furthermore, the last syllable should be extended by 20%. Finally, we must define the transition from node to node:

```
render(n1, n2, "log");
render(n2, n5, "fujisaki");
render(n3, n4, "log");
render(n4, n5, "log");
render(n2, n3, "breath");
```

The above script says that nodes n1 and n2 should be connected by an exponential function in the "syllable" level. Then, nodes n2 and n5 should be connected using the Fujisaki's intonation model (declination). After that, we apply the inverted pitch to node n4 and finally, we apply a breathing algorithm between n2 and n3.

5. Discussion

The major offer of DEMOSTHeNES Speech Composer, is the option to mix text and instructions in an open environment. This scheme allows a more functional communication among modules in the system, so that processing is distributed to the corresponding modules. For example, in the Greek language numbers usually refer to nouns and thus they should acquire the grammatical state of the corresponding nouns. Using the approach presented here, the Finite State Transducer can identify and convert the number to the corresponding string (masculine, singular, nominative) and further wrap the string with an instruction that indicates its numerical nature. A converter VMOD can then examine the following and the previous word (simple case) using POS information and convert the endings of the numerical string to the appropriate declension.

DEMOSTHeNES Speech Composer has been implemented in MS-Win32 platform, however, the kernel and most of the VCOM are portable to other platforms as well. It has been wrapped to an ActiveX control that allows it to be embedded to other applications. This control can have either a very simple interface to be easily used from other applications that require basic TtS functionality (speak, stop), or a full

MS-SAPI support for more complex manipulation of the provided functionality.

There is currently a configuration (i.e. a set of VMODs in a specific chain) for supporting the Greek language and partially the English one. This configuration forms a general purpose TtS system, that is quite powerful in handling text patterns (more than 800 acronyms and abbreviations, in any declension, several date, hours and numeric patterns are supported using the FST VCOM), offers an enriched prosody compared with other commercial systems and is bundled with an new MBROLA voice (gr2). CART trees has been used in predicting duration values in phoneme level, while the rule-based approach presented in §4.2 is being used for intonation rendering. Prosodic Templates currently covers all the major marks (Level A), conjunctive phrases (Level B) and some categories of Level C. The English language is supported by using a CART version of the CMU dictionary that is distributed with the FESTIVAL system.

The configuration of DEMOSTHeNES for supporting the Greek and the English language in polyglot environments showed the flexibility of the system and its ability to reuse existing resources. Also, the modular architecture constituted a very flexible experimental environment that allowed the comparison of different approaches in several issues (like rule-based duration vs. trained duration).

6. Conclusion

In this paper, we presented in detail the architecture of DEMOSTHeNES Speech Composer and examples of text handling and prosody generation. This architecture takes advantage from existing tools and furthermore extends the ordinary architecture of TtS systems to accommodate a scheme for describing in detail the functionality of the system under specific circumstances. DEMOSTHeNES currently fully supports the Greek language and partially the English and can be downloaded from [5].

7. Acknowledgments

The work described in this paper has been partially supported by the M-PIRO project of the IST Programme of the European Union under contract no IST-1999-10982.

References

1̃ Black, A. and Taylor, P.: *CHATR: a generic speech synthesis system*, COLING94, II pp 983-986, Kyoto, Japan, 1994

2̃ Taylor, P., Black, A. and Caley, R., *The architecture of the Festival Speech Synthesis System*, 3rd ESCA Workshop on Speech Synthesis, Jenolan Caves, Australia pp. 147-151, 1998

3̃ Dutoit, T., Bagein, M., Malfrere, F., Pagel, V., Ruelle, A., Tounsi, N. and Wynsberghe, D., *EULER : an Open, Generic, Multi-lingual and Multi-Platform Text-To-Speech System*, In Proceedings of LREC'00, Athens, Greece, pp. 563-566, 2000

4̃ Taylor, P., Black, A. and Caley, R, *Heterogeneous relation graphs as a formalism for representing linguistic information*, Speech Communication, Volume 33, Nos. 1-2, pp 153-174, January 2001

5̃ DEMOSTHeNES Speech Composer homepage http://www.di.uoa.gr/speech/synthesis/demosthenes

6̃ Xydas, G. and Kouroupetroglou, G.: *Augmented Auditory Representation of e-Texts for Text-to-Speech Systems*, in Proceedings of the 4th International Conference on Text, Speech and Dialogue, TSD 2001, Plzen (Pilsen), Czech Republic, pp 134-141, September 2001

7̃ Xydas, G. and Kouroupetroglou, G.: *Text-to-Speech Scripting Interface for Appropriate Vocalisation of e-Texts*, in Proceedings of the 7th European Conference on Speech Communication and Technology, EUROSPEECH 2001, Aalborg, Denmark, pp 2247-2250, September 2001

8̃ *Voice eXtensible Markup Language (VoiceXML$^{\text{TM}}$) version 1.0*, W3C Note 05 May 2000, http://www.w3.org/TR/voicexml/

9̃ The FESTIVAL Speech Synthesis System homepage http://www.cstr.ed.ac.uk/projects/festival/

10̃ *XSL Transformations (XSLT) Version 1.0*, W3C Recommendation 16 November 1999, http://www.w3.org/TR/xslt