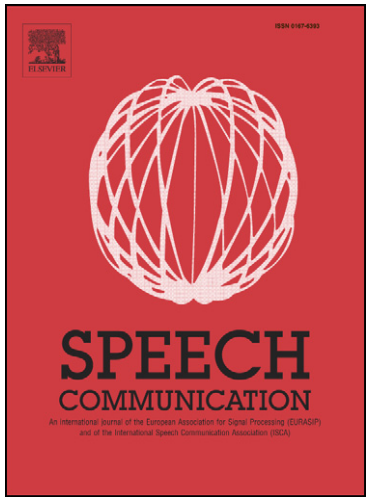
Time and Space-Efficient Architecture for a Corpus-based Text-to-Speech Synthesis System

Matej Rojc, Zdravko Kačič



Time and Space-Efficient Architecture for a Corpus-based Text-to-Speech Synthesis System

Matej Rojc, Zdravko Kačič

PII: S0167-6393(07)00022-2
DOI: 10.1016/j.specom.2007.01.007
Reference: SPECOM 1612

To appear in: *Speech Communication*

Received Date: 7 June 2006
Revised Date: 21 January 2007
Accepted Date: 23 January 2007

Please cite this article as: Rojc, M., Kačič, Z., Time and Space-Efficient Architecture for a Corpus-based Text-to-Speech Synthesis System, *Speech Communication* (2007), doi: 10.1016/j.specom.2007.01.007

Title:

Time and Space-Efficient Architecture for a Corpus-based Text-to-Speech Synthesis System

Authors:

Matej Rojc
Zdravko Kačič

Affiliation:

Faculty of Electrical Engineering and Computer Science, University of Maribor
Smetanova ulica 17
2000 Maribor
Slovenia

Mail:

matej.rojc@uni-mb.si
kacic@uni-mb.si

Corresponding author:

Doc.dr. Matej Rojc
Faculty of Electrical Engineering and Computer Science, University of Maribor
Smetanova ulica 17
2000 Maribor
Slovenia
Mail: matej.rojc@uni-mb.si
Telephone: +386 2 220 7223
+386 31 879 696
Fax: +386 2 251 11 78

Abstract:

This paper proposes a time and space efficient architecture for a text-to-speech synthesis system (TTS). The proposed architecture can be efficiently used in those applications with unlimited domain, requiring multilingual or polyglot functionality. The integration of a queuing mechanism, heterogeneous graphs and finite-state machines gives a powerful, reliable and easily maintainable architecture for the TTS system. Flexible and language-independent frameworks efficiently integrate all those algorithms used within the scope of the TTS system. Heterogeneous relation graphs are used for linguistic information representation and feature construction. Finite-state machines are used for time and space efficient representation of language resources, for time and space efficient lookup processes, and the separation of language-dependent resources from a language-independent TTS engine. Its queuing mechanism consists of several dequeue data structures and is responsible for the activation of all those TTS engine modules having to process the input text. In the proposed architecture, all modules use the same data structure for gathering linguistic information about input text. All input and output formats are compatible, the structure is modular and interchangeable, it is easily maintainable and object oriented. The proposed architecture was successfully used when implementing the Slovenian PLATTOS corpus-based TTS system, as presented in this paper.

Keywords:

Corpus-based text-to-speech system, finite-state machines (FSM), heterogeneous-relation graphs (HRG), queuing mechanism

Time and Space-Efficient Architecture for a Corpus-based Text-to-Speech Synthesis System

Matej Rojc, Zdravko Kačič
Faculty of Electrical Engineering and Computer Science, University of Maribor

Abstract:

This paper proposes a time and space efficient architecture for a multilingual text-to-speech synthesis system (TTS). The proposed architecture can be efficiently used in those applications with unlimited domain, requiring multilingual or polyglot functionality. The integration of a queuing mechanism, heterogeneous graphs and finite-state machines gives a powerful, reliable and easily maintainable architecture for the TTS system. Flexible and language-independent frameworks efficiently integrate all algorithms used within the scope of the TTS system. Heterogeneous relation graphs are used for linguistic information representation and feature construction. Finite-state machines are used for time and space efficient representation of language resources, for time and space efficient lookup processes, and the separation of language-dependent resources from a language-independent TTS engine. Its queuing mechanism consists of several dequeue data structures and is responsible for the activation of all those TTS engine modules having to process the input text. In the proposed architecture, all modules use the same data structure for gathering linguistic information about input text. All input and output formats are compatible, the structure is modular and interchangeable, it is easily maintainable and object oriented. The proposed architecture was successfully used when implementing the Slovenian PLATTOS corpus-based TTS system, as presented in this paper.

1. Introduction

A lot of TTS systems have been developed around the world over the last decade (Campbell and Black, 1996; Syrdal et al., 2000; Taylor et al., 1998; Holzapfel, 2000; Sproat, 1998). TTS systems consist of several processing steps and many of them are more or less language-dependent. Various applications in the field of speech technology need more and more multilingual and polyglot TTS architectures that have to be time and space efficient. In order to meet such requirements, the use of separate programs for each processing step, different input and output formats, different data structures for different tasks etc. are certainly undesirable. Due to these facts and due to the need for a powerful, reliable and easily maintainable text-to-speech synthesis system a design pattern needs to be developed that would serve as a flexible and language independent framework for pipelining text-to-speech processing steps. In order to meet these goals, two TTS systems were of particular interest when developing the proposed TTS architecture. Namely, both systems contain data structures that make them efficient and flexible. The Lucent TTS system (Sproat, 1998) is based on finite-state machines and the Festival system (Clark et al., 2004) is based on heterogeneous relation-graph structure (Taylor et al., 2001). Finite-state machines are very interesting because of the various linguistic processing issues found in the TTS system (Sproat, 1998) and heterogeneous relation graphs represent flexible formalism for the representation of linguistic information gathered from input text (Taylor et al., 2001). Our goal was to develop architecture that would benefit from both data structures, would be easily maintainable and would allow flexible migration to new languages, have efficient data flow through the whole system and between modules, and allow simple monitoring and performance evaluation after each module.

The presented paper proposes a new architecture for the TTS system, where finite-state machines and heterogeneous relation graphs are integrated into a common TTS engine through the so-called “queuing mechanism”. In this way all text-to-speech processing modules are pipelined together. The finite-state machines are a time-and-space efficient representation of language resources and are used for the separation of language-dependent parts from the language-independent TTS engine. Heterogeneous relation graphs that store all the knowledge about each input sentence are used for the representation of, linguistically, very heterogeneous data and for those complex feature constructions needed by various machine-learned models used in the TTS system.

In the presented approach, all the algorithms in the TTS system use the same data structure for gathering linguistic information about input text, all input and output formats between modules are compatible, the structure is modular and interchangeable, easily maintainable and object oriented. The proposed TTS architecture integrates into a common TTS engine, all the processing steps that are usually found in a TTS system: tokenisation, part-of-speech tagging, grapheme-to-phoneme conversion, symbolic and acoustic prosody, unit selection, concatenation, and acoustic processing. These processing levels are also considered in the description of the proposed architecture. It is easy to add additional ones or remove some of them from the architecture. This novel architecture itself, and its implementation in the Slovenian corpus-based PLATTOS TTS system, will be presented in the rest of this paper.

The general architecture of the corpus-based TTS system is presented in section 2. All data structures used in the proposed architecture are described in the next section. Here, the reasons for implementation of selected data structures are discussed in more detail. Section 4 presents the proposed architecture of the TTS system, and each single module is described. In section 5, the implementation of the proposed architecture for the Slovenian PLATTOS corpus-based TTS system is described in detail. A conclusion is drawn at the end.

2. General Architecture of the TTS system

In the general architecture of any TTS system (Figure 1), the following modules are normally used: tokenizer, morphology analysis, part-of-speech tagger (POS), grapheme-to-phoneme conversion, symbolic and acoustic prosody module, unit selection module, concatenation, and acoustic module. Various language knowledge resources, e.g. phonetic and morphological lexica, linguistic rules, and acoustic speech database, can be used as the external language-dependent part. In the tokenizer module the input text sentence is first broken into tokens. Abbreviations, special symbols, and numbers must be converted into their corresponding word forms (Sproat, 1998). Then morphological analysis of the tokens is performed, usually assigning more than one POS tag to the tokens. By using a part-of-speech tagger in the next module, only one part-of-speech tag is assigned to each token, after considering the context (Brill, 1993). In the grapheme-to-phoneme conversion module, the phonetic transcriptions are assigned to the words. The prosody modules follow. In some systems the symbolic and acoustic modules are merged, in others they are separated. Normally, in a symbolic prosody module, phrase breaks, prominence, and intonation patterns are usually predicted and assigned to the sentence. The acoustic prosody module defines the segment durations, pause durations and F0 contours. The unit selection module uses acoustic inventory, which is constructed during the processing of the speech database (usually found in any corpus-based TTS system). The acoustic inventory contains those unit candidates that are used for the conversion of input text into speech signal. Unit candidates must be found that are as close as possible to the desired prosody, as predicted by the prosody modules (Bulyko, 2001; Bulyko and Ostendorf, 2001). This task is a big issue, especially in the case of corpus-based TTS systems, regarding time and space efficiency. In the acoustic processing module, the concatenation points are processed and a speech synthesis algorithm such as PSOLA is usually used for generating the final speech signal (Sproat, 1998; Holzapfel, 2000).

Linguistic information must be efficiently and flexibly stored in all TTS systems. In addition, linguistic data processed in a TTS system are very heterogeneous. TTS systems are involved in text analysis, syntactic analysis, morphology, phonology, phonetics, prosody, articulatory control, and acoustics. Therefore, it is highly desirable that different types of linguistic information use a single formalism. All modules in the system need external language resources. These must be efficiently separated from the system, in order to have a language-independent TTS engine. On the other hand, the access to language resources must be fast, in order to meet real-time requirements, and their representation must be time and space efficient. These are also the main issues that have been considered in the proposed TTS architecture.

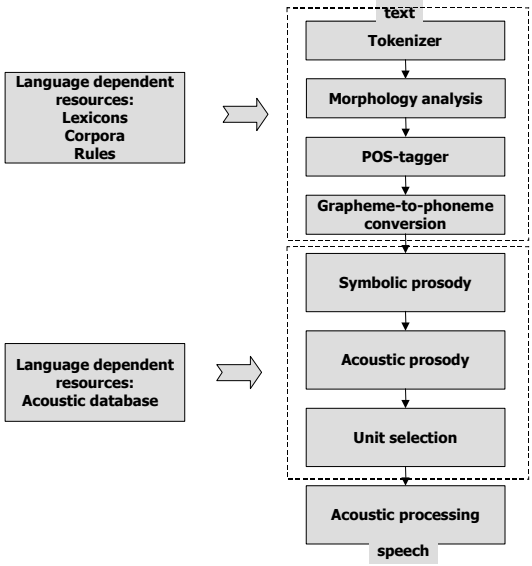


Figure 1: General architecture of the TTS system.

3. Data structures used in the proposed TTS architecture

Three data structures are used in the proposed architecture: dequeues, heterogeneous relation-graphs and finite-state machines. Dequeues can be used for the construction of flexible and language-independent queuing mechanisms that integrate all the modules in the TTS system (Horowitz and Sahni, 1996; Holzapfel, 2000). Heterogeneous relation graphs can be used for storing linguistic and acoustic information that is extracted from input texts. They can also be used for the very flexible and transparent construction of complex features needed by some machine-learned models used in the system (Taylor et al., 2001). Finite-state machines can be used for time and space-efficient representation of external language-dependent resources and linguistic rules (Sproat, 1998; Mohri, 1995). They can also provide a general mechanism for the separation of language-dependent resources from a language-independent TTS engine. All these data structures have been used because their features were found to be very useful for solving many architectural and performance issues. These features will be briefly outlined in the following sections.

3.1. Queue

A queue data structure is an ordered collection of items, from which items may be deleted at one end (front of the queue) and into which items may be inserted at the other (rear of the queue), (Horowitz and Sahni, 1996). An important usage of queues is input/output buffering. Clearly, the queue must be organized as a first-in-first out structure. An empty queue condition indicates that the input buffer is empty and one module execution is suspended, while the previous module loads more data into the buffer. Such a buffer has a limited size, and thus a queue-full condition must also be used to signal when it is full and no more data is to be transferred. The insert and delete operations are restricted, so that insertions are performed at only one end, and deletions at the other. In the proposed architecture, the insertions and deletions must be made at both ends. To model these situations, a double-ended queue, abbreviated as dequeue, is the proposed data structure for use. A dequeue is found to be an appropriate data structure for modelling the processes found in the TTS engine, where modules process the input sentence successively.

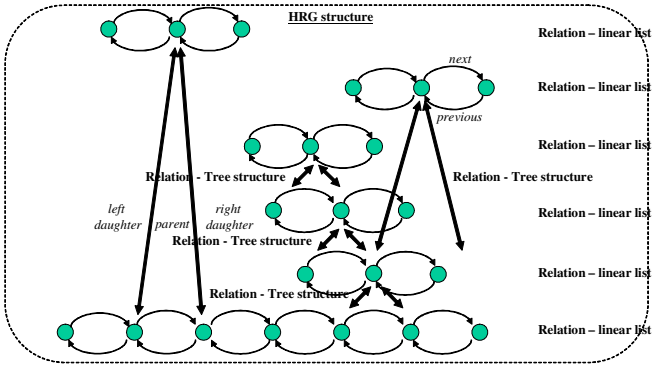


Figure 2: A general HRG structure.

3.2. Heterogeneous relation graph (HRG)

A heterogeneous Relation Graph (HRG) is a formalism for describing linguistic structures (Taylor et al., 2001). It was developed for use in speech synthesis systems, therefore, its design reflects the specific needs of speech synthesis systems. HRG can be used to store any type of linguistic data. The HRG formalism can represent the required input and output information for any processing module found in the system, irrespective of what type of process the system's module is involved in, or the methods or algorithms are used. A major additional requirement is that access to the stored information is fast, since the speech synthesis system must be both fast and efficient. On the other hand, the formalism must still provide a clean general-purpose mechanism for storing all the linguistic information. In speech synthesis systems the structures are usually not static descriptions of an utterances' lingustic contents. They usually contain incomplete linguistic content of the utterance in the middle of the processing stage, converting one piece of information into another, etc. Therefore, linguistic information in the structures is often added, removed or enriched during processing. Such operations must be performed flexibly and safely. Any replications of information must also be avoided in such linguistic structures. All common types of linguistic structure must be accommodated by including lists, trees and autosegmental diagrams (multi-linear structures), (Taylor et al., 2001). The relations between different kinds of linguistic information must be efficiently represented, often in the form of some hierarchical structure.

In the TTS system, the HRG structure consists of linguistic objects that can be, e.g. words, syllables etc. and are represented by objects - *linguistic items*. All these items exist in the so-called »*relation structures*«, which specify the relations between the items. A relation structure exists for each required linguistic type. A heterogeneous relation graph contains all the relations and items that are specified for current utterances. Furthermore, relations are structures that are composed of nodes and named arcs. Nodes don't contain any information, they just serve as positional units, which point to linguistic items. Additionally, relations can be trees, linear lists, multi-linear structures or even other structures. All arcs in a HRG structure occur in complimentary named pairs, or as acyclic graphs, in which each arc has two complimentary names. A node may have any number of arcs. The names of the arcs in the outgoing direction must be unique, whereas the names of the arcs in the incoming direction do not have to be unique. A general HRG structure containing linear lists and tree structures, used for storing linguistic information, is presented in Figure 2. As can be seen from Figure 2, both types of relations consist of three components: items, nodes and arcs. These items contain the linguistic information. The nodes and arcs define the relations between the items. In the tree type of relations, the arcs occur in complementary pairs named *parent* and *daughter1*, *daughter2* etc. Here, a node in the HRG structure can have any number of arcs.

Items in the HRG structure are attributed value lists (AVL), which contain linguistic information. Items are usually composed of two types of AVL, called *Contents* and *Relations* (Taylor et al., 2001). The contents part contains infomation such as the local linguistics information of the item, and the relation part specifies which nodes in which relations are linked to this item. The relations part consists of an AVL of all the relations that item is in. The attribute is the name of the relation and the value is the node in the relation that the item is linked to. Nodes are described by feature structures by having an attribute for each named arc leaving the node, and a single attribute item whose value is the item. Therefore, an entire HRG structure in the proposed architecture can also be represented as a feature structure. In the TTS engine, algorithms in the queuing mechanism work constantly on some subsets of the HRG structure. Therefore, it is necessary to access these subsets as efficiently as possible. The TTS engine accesses all the linguistic information stored in the HRG structure, relating to one or more linguistic types: e.g. all the segments and all the syllables. Accessing items of a given linguistic type is extremely efficient. Since the nodes, arcs and items are explicitly separated, the HRG structure ensures better maintainability and less possibility for the construction of a corrupted linguistic structure.

In the proposed architecture, HRG structures are used for representation of the heterogeneous and dynamically changing linguistic information extracted from the input text. The formalism can also be used for the construction of those complex linguistic features needed by trained CART prediction models, when used in the TTS system.

3.3. Finite-state machines (FSM)

Finite-state machines (FSM) represent an attractive solution for many linguistic processing issues found in the TTS systems, since they have the following interesting features (Mohri, 1995; Arnaud, 2000):

- optimal speed: matching a string with a deterministic finite-state machine, takes a time proportional to the input size and is independent on the size of the finite-state machine
- modularity: various linguistic objects can interact by using operations defined on finite-state devices. Cascades of transducers can model complex relations from simpler ones
- compactness: e.g. the lexicons' representations decrease their size
- easy processing: construction of finite-state machines with elementary operations close to those of the sets. Complexity of operations is well defined and, therefore, makes them appropriate for real time computations
- large-scale optimisation: many efficient minimization algorithms exist
- software design: the overall system is less error prone

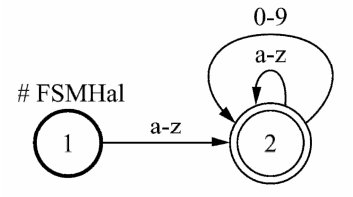


Figure 3: A general finite-state automaton (detection of word tokens).

Finite-state automata (FSA) can be seen as a directed graph with labels on each arc (Figure 3). A finite-state automaton A is a 5-tuple (Σ, Q, i, F, E), where Σ is a finite set called the alphabet, Q is a finite set of states, $i \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $E \subseteq Q\mathrm{x}(\Sigma \cup \{\in\})\mathrm{x}Q$ is the set of edges. FSAs are shown to be closed under union,

Kleen star, concatenation, intersection and complementation, thus allowing for natural and flexible descriptions. They are also very flexible due to their closure properties (Mohri, 1995).

Finite-state transducers (FST) can be interpreted as defining a class of graphs, a class of relations on strings, or a class of transductions on strings. In the first interpretation, an FST can be seen as an FSA, in which each arc is labelled by a pair of symbols rather than by a single symbol. A finite-state transducer T is a 6-tuple $(\Sigma_1, \Sigma_{2,} Q, i, F, E)$, where Σ_1 is a finite alphabet, namely the input alphabet, Σ_2 is a finite alphabet, namely the output alphabet, Q is a finite set of states, $i \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $E \subseteq Q \mathrm{x} \Sigma_1^* \mathrm{x} \Sigma_2^* \mathrm{x} Q$ is the set of edges. As with FSAs, FSTs are also powerful because of their various closure and algorithmic properties (Mohri, 1995).

Finite-state machines have been used in various domains of natural language processing. Finite-state transducers that output weights, also play an important role in language and speech processing. Their use can be justified by linguistic and computational arguments (Mohri, 1997). From the linguistic point of view, they often lead to a compact representation of lexical rules, idioms etc. Graphic tools also allow us to visualise and modify constructed machines. The latter is usually helpful in grammar constructions. From the computational point of view, the use of finite-state machines is mainly motivated by considerations of time and space efficiency. Time efficiency is usually achieved by determinization. The output of deterministic machines, in general depends linearly only on the input size and can, therefore, be considered as optimal from this point of view (Mohri, 1997). Space efficiency is achieved using classical minimization algorithms (Aho, Hopcroft, and Ullman, 1974) for deterministic automata. The applications in natural language processing, which range from the construction of lexical analyzers (Silberztein, 1993) and the compilation of morphological and phonological rules (Kaplan and Kay, 1994; Karttunen, Kaplan and Zaenen, 1992) to speech processing (Mohri, Pereira, and Riley, 1996) show the usefulness of finite-state machines.

Time and space efficiency is very important when dealing with language. The size of language resources regarding time and space efficiency is an important issue in the case of corpus-based TTS systems. In order to solve this problem, sequential finite-state transducers are generally used (string-to-string transducers, string-to-weight transducers). Sequential transducers are transducers with a deterministic input. In any state of such transducers, at most one outgoing arc is labeled with a given element of the alphabet. Output labels might be strings, including the empty string ε. They are computationally very interesting because their use with a given input does not depend on the size of the transducer but only on that of the input. Since that use consists of following the only path corresponding to the input string and writing consecutive output labels along this path, the total computational time is linear in the size of the input, when the cost of copying out each output label does not depend on its length. They have been successfully used in the representation of large-scale dictionaries, computational morphology, local grammars, syntax etc. For representation of language models, phone lattices, and word lattices, string-to-weight transducers can be used. Algorithms for determinisation and minimization of the sequential transducers were defined in Mohri, 1997. The minimization of sequential string-to-weight transducers can also be performed using the determinization algorithm (Mohri, 1997). Most of these algorithms have been used in speech processing. The determinization and the minimization algorithms can be used to reduce the size of the transducers. The composition, union, and equivalence algorithms for sequential transducers are also very useful in many speech processing applications (Mohri, 1997).

In the proposed architecture, FSMs are used for the separation of language dependent linguistic resources from the language independent TTS engine. FSMs are also used for time and space efficient representation of all linguistic resources used in the system.

4. The proposed architecture of the TTS system

The proposed architecture is modular, time and space efficient, and flexible. The language dependent resources are separated from the language independent TTS engine, using FSM formalism. All modules in the system are easy to maintain and improve. The modules allow easy integration of new algorithms into the common queuing mechanism used in the TTS system.

4.1 Queuing mechanism used in the architecture

In Figure 4, part of the queuing mechanism, composed from several double linked lists - dequeues, is presented as the first data structure used in the proposed architecture. Each double-linked list (dequeue) is used for one processing step in the TTS system. All dequeues are linked together into a flexible mechanism, used for pipelining all TTS processing steps together.

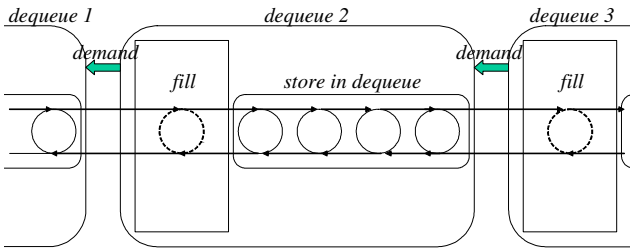


Figure 4: The queuing mechanism composed of several dequeues connected together.

The following dequeues are proposed for use in the TTS architecture, representing modules already defined in the general architecture of the TTS system in Figure 1:

- tokenization dequeue
- part-of-speech tagging dequeue
- grapheme-to-phoneme conversion dequeue
- symbolic prosody dequeue
- acoustic prosody dequeue
- unit-selection dequeue
- concatenation dequeue and
- acoustic dequeue

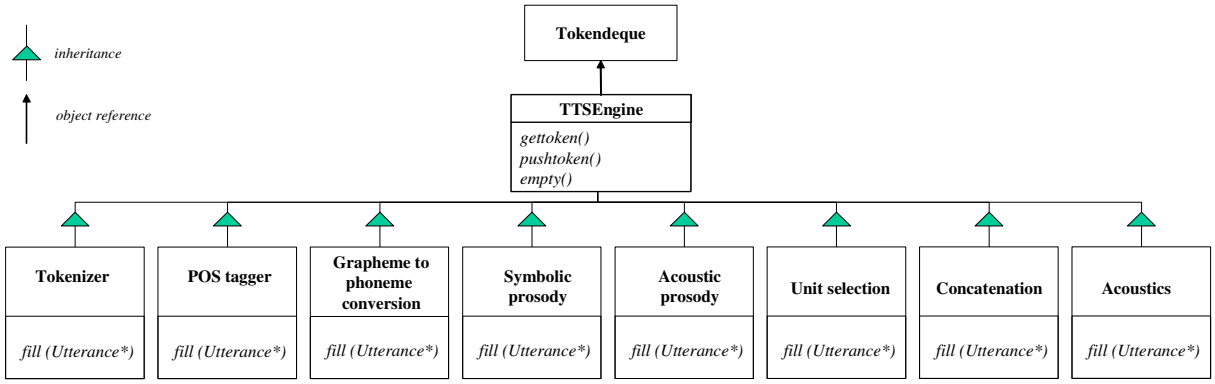


Figure 5: A class hierarchy used for queuing mechanism construction.

Each dequeue corresponds to each module of the TTS engine. In order to construct a queuing mechanism, a class hierarchy is constructed from an abstract base class named 'TTSEngine', as shown in Figure 5. The queuing mechanism is constructed in the initialisation process. The TTS engine objects in Figure 5 are created, and a reference on the dequeue of the next object in the TTS engine is also assigned to each one. The main process runs in a loop, when processing the input text. As seen in Figure 6, the queuing mechanism starts with the main function using reference to the acoustic's module object. The main process sends demand to the acoustic module by using '*gettoken()*' function. At the start all the dequeues of the TTS engines' objects are empty (*empty()* function returns true condition). Therefore, the demands with '*gettoken()*' function that are performed in '*fill()*' function, travel from the acoustic module to the tokenizer module. In Figure 6 it can be seen that the tokenizer module then generates tokens by using a scanner (usually based on finite-state machines). Tokens can be of various types: e.g. punctuations, words, acronyms, cardinal, ordinal and float numbers etc. Two additional token types are added for marking 'end of sentence' (EOS) and 'end of file' (EOF) conditions. These tokens are solely used for controlling the queuing mechanism. When the POS tagging module accepts an EOS token from the tokenizer module, it performs tagging. This module needs all the sentence tokens, since tagging usually works at sentence level. After tagging, the grapheme-to-phoneme conversion module grabs tokens from the POS tagging dequeue by using demand '*gettoken()*', until an EOS token is accepted. When grapheme-to-phoneme conversion is done, all sentence tokens are pushed to symbolic prosody dequeue. This process continues until acoustics dequeue, where the speech signal for the corresponding sentence is generated. POS tagging, grapheme-to-phoneme conversion, symbolic and acoustic prosody, unit selection, concatenation and acoustics usually work at sentence level, therefore, all sentence tokens are processed at once and pushed to the next dequeue. When acoustic processing for a given sentence is finished, the main process has to check whether the 'end of file' condition token is in the acoustic dequeue. If not, the

process releases the memory used by the tokens and continues to process the next sentence in the same way as before. Otherwise, the memory is released and the main process-loop is stopped. Demands that can be performed by TTS engine objects are closely connected with double linked queue features: *gettoken()*, *pushtoken()* etc. *Fill()* function is able to pass through tokens, modify them, insert new tokens or delete existing ones. These demands and features of the dequeues can be helpful for the developer of the TTS system, when developing new algorithms in corresponding modules.

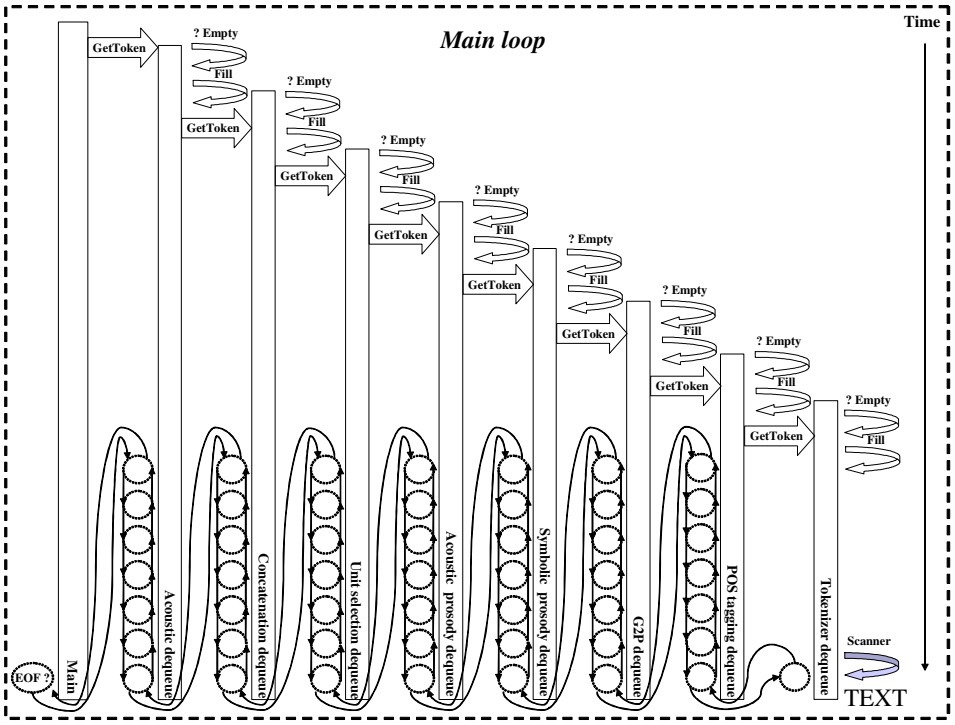


Figure 6: Interaction between dequeues in the queuing mechanism.

The presented queuing mechanism is also flexible. It is possible to break the mechanism at any TTS engine object, or add new ones. Therefore, the results after each module are easily observable and monitored. This can be quite helpful, for example, when testing specific algorithm performance or doing resynthesis. The mechanism can also be quickly adapted for example (simple changes in *fill*() function), when processing in a module does not need all sentence tokens. The system granularity at the token level can be used, for example using language models (LM) for EOS detection, doing normalisation, processing of whitespaces, end-of-line tokens or morphology analysis. All higher modules usually process sentences and are activated after obtaining all sentence tokens. Nevertheless, the granularity of the system can be changed easily, when desired. Actual tokens are moved from module to module, since status of the dequeue (empty, full, number of tokens etc.) is used for running the queuing mechanism – TTS engine. Therefore, after the main process also finds an EOS token in the acoustic dequeue, the queuing mechanims knows that the tokens can finally be removed from the queuing mechanism.

The TTS processing steps are sequential (e.g. one step cannot be started before the previous one). But it makes sense that during the processing of one sentence, the processing of the next could already be started and the corresponding processing be running. Threads can be used in this case, since they can share the resources of the parent process. No additional preparative initialisation is needed. After creation, threads are independent entities from their parents. It is also possible that there is one parent queuing mechanism process and that algorithms in the TTS engine modules run in separate threads. In this case proper management of the threads has to be carried out. The creator can also change the priority of threads. Current sentence processing should have higher priority in this case.

4.2 Heterogeneous relation graphs used in the architecture

All modules in the speech synthesis system need an efficient and flexible formalism for describing those linguistic structures found in the input text. The HRG structure can store any type of linguistic information extracted from the input text, in a flexible way (Taylor et al., 2001). HRG structure also provides clean general-purpose mechanisms for

representing the linguistic information extracted by the TTS system. All TTS modules contribute to the linguistic information used for generating the speech signal.

In the proposed architecture, one HRG structure is used for this purpose and is made accessible by all modules in the TTS system, where reference to HRG structure is represented as an argument of the *fill()* function (*Utterance**) as shown in Figure 5. Therefore, algorithms in given modules are able to access, change or enrich stored linguistic information when appropriate. The used HRG structure is not static and can dynamically change through the queuing mechanism, e.g. contained linguistic information or the HRG structure itself, since TTS modules are adding relation structures during processing. These structures can be in the form of linear lists or in the form of trees, depending on the type of linguistic information that should be stored in the HRG structure by the corresponding module. HRG structure represents, in the final TTS module (acoustic module), all the linguistic information extracted from the input sentence. It is used finally for generation of the speech signal. Figure 7 illustrates the integration of the queuing mechanism and heterogeneous relation-graph. The HRG structure demonstrates the use of two types of proposed relation structures, regarding the linguistic information processing in the TTS engine, in the forms of linear lists and trees. As seen in Figure 7, the relation structures in the form of linear lists are named *Segment, Syllable*, *Word*, *Phrase*, *IntEvent*, and *SynUnits*. The relation structures in the form of trees are named *SyllableStructure*, *PhraseStructure*, *IntonationStructure*, *SynUnitsStructure*. The names of the linear lists are assigned according to the linguistic objects they are storing. The linguistic objects in the normally used TTS systems can be:

- words (*Word* relation structure – POS tagging module)
- syllables, segments (*Segment* and *Syllable* relation structures – grapheme-to-phoneme conversion module)
- phrase breaks, intonation events (*Phrase* and *IntEvent* relation structures – prosody modules)
- synthesis units (*SynUnits* relation structure – unit selection, concatenation and acoustic modules)

These linguistic objects are represented in the HRG structure by objects termed as *linguistic items* (Figure 7). All these items are elements of relation structures named *Segment*, *Word*, *Syllable*, *Phrase*, *IntEvent*, *SynUnits*, in the form of linear lists. Linear lists are able to specify the relation between all items found in the input sentence. In this case the arcs occur in complementary pairs named *next* and *previous*. Therefore, forward and backward traversals in the structure are possible. As a result, the HRG graph contains all the relations between items in a given linear list for the current sentence. As seen in Figure 7, additional relation structures in the form of trees are added. Namely, some TTS engine modules can use machine-learned models (CART trees, neural networks etc.) for example the prediction of prosody parameters in the input sentence. These modules need complex feature vectors. The tree relation structures add vertical information between linguistic objects in different linear lists. Therefore, much more complex features can easily be obtained from the constructed HRG structure. The *SynUnitsStructure* relation structure relates those synthesis units' items found in the *SynUnits* relation structure and the phoneme segments' items in the *Segment* relation structure. The *IntonationStructure* relation structure relates intonation events' items in the *IntEvent* relation structure and syllable items in the *Syllable* relation structure. The *SyllableStructure* relates segments' items in the *Segment* relation structure and syllable items in *Syllable* relation structure. The same name of tree relation structure is also used for linking syllable items in *Syllable* relation structure and word items in *Word* relation structure. The *IntonationStructure* relates intonation event items in the *IntEvent* relation structure and syllable items in the *Syllable* relation structure (Rojc, 2003). Naturally, the relation structures used in Figure 7, can easily be changed and adapted to different structures, following the processing needs of the modules in the TTS engine.

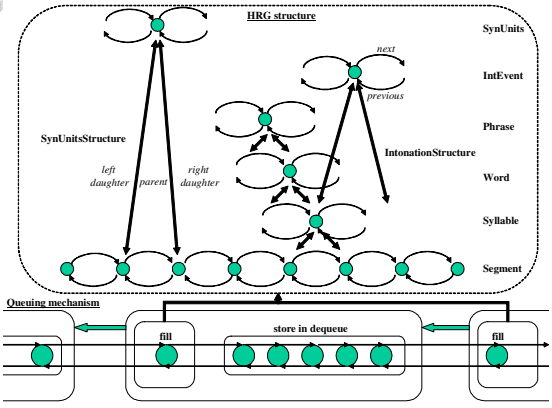


Figure 7: Integration of a queuing mechanism with a heterogeneous relation graph.

Another issue in the TTS system is the flexible and efficient construction of features that are needed for machine-learned models, e.g. CART trees. HRG structure is flexible representation of linguistic information extracted by TTS modules in the TTS engine. It is possible to construct complex linguistic and acoustic features for machine-trained models (CART trees) quickly and easily. It is just a matter of picking-up those values in the items accessed through defined relations in the structure, without any additional processing or extra work on feature construction. All atomic linguistic entities such as segments, syllables, words, phrases, intonation events, unit candidates etc. are represented by items. Attributes are the named linguistic properties defined in the TTS system modules, e.g. part-of-speech, duration, phone class and properties, intonation event type, phrase break type, prominence label type etc. Values can be strings, enumerated sets, floating point numbers and integers. Typical word and segment items are shown in Figure 8.

pos	Mcmpnl
word	Dvesto
phrase break	0

segment	d
duration	90.6ms
class	plosiv
stress	0
syllbreak	0
endofword	0

Figure 8: Example for typical word item '*Dvesto/two hundred*' (left) and segment item'd' (right).

When the main process finds the EOS token in the acoustic dequeue, the queuing mechanism recognises that the tokens can be finally removed from the queuing mechanism. At this time the corresponding HRG structure can also be deleted (emptied) and a new one created, when there are new sentences to be processed.

4.3 Finite-state machines used in the architecture

In the development of multilingual and polyglot speech synthesis systems, it is very important that the migration to new language can be done as flexibly as possible and with little or no intervention in the algorithms used in the TTS engine. Clearly, this can be achieved by the development of such algorithms, which perform separation of language-dependent language resources from the TTS engine. A language-independent TTS engine is obtained, when this is achieved. Such a TTS engine simplifies migration to new languages. The efficient separation of language-dependent language resources can be done by finite-state machines (Mohri, 1995; Watson, 1995; Daciuk, 1998). Finite-state machines are also very appropriate for the representation of language resources and linguistic rules. They can be constructed previously off-line and simply loaded into the TTS engine during on-line operation, e.g. very large-scale dictionaries can be represented by finite-state transducers. The corresponding representation then offers fast look-up, since the recognition does not depend on the size of the dictionary but only on the length of the considered input string. The minimization algorithm for sequential transducers allows one to reduce to a minimum the sizes of these devices. Experiments have shown that one can obtain, in an efficient way, compact and fast look-up representations for large natural language dictionaries (Mohri, 1997). Context-dependent phonological and morphological rules can be represented by finite-state transducers (Kaplan and Kay, 1994). The result of the computation described by Kaplan and Kay (1994) can be determinized, increasing considerably the time efficiency of the transducer. It can be further minimized to reduce its size. These observations can be extended to the case of weighted rewrite rules (Mohri and Sproat, 1996). All levels of the text-to-speech synthesis system without acoustic processing level can be represented by a composition of finite-state transducers outputting the sequence of selected units. Nevertheless, due to the size of the obtained machine, a sequence of smaller machines can result in a more efficient solution. In the TTS system engine, finite-state machines can be used, as follows:

- tokenizer (constructed from regular expression by FSM compiler)
- spell checking system (constituent part of the text normalisation)
- normalisation of abbreviations, acronyms, numbers, and special symbols
- part-of-speech tagging
- grapheme-to-phoneme conversion
- foreign word detection, unit selection search algorithm etc.

In some modules, machine-learned models must be used, e.g. for the prediction of phrase breaks, prominence, intonation event labels etc. The proposed architecture suggests that decision trees should be used as prediction models, but in general other machine-learned models could also be used. Decision trees provide already space efficient knowledge representation. They can also be compiled into weighted finite-state transducers, as merging with other finite-state

machines used in modules results in an improved performance and flexibility of the whole system (Sproat and Riley, 1996; Mohri and Sproat, 1996).

4.4 The TTS engine

Figure 9 shows the proposed architecture of the TTS system, separated into language-dependent language resources and a language-independent TTS engine. The structure is composed of heterogeneous relation graphs, a queuing mechanism, and finite-state machines. The heterogeneous relation graph gathers linguistic data for the corresponding sentence extracted by modules of the TTS engine. The queuing mechanism consists of several dequeues for tokenizing, POS tagging, grapheme-to-phoneme conversion, symbolic prosody and acoustic prosody processing, unit selection, concatenation and acoustic processing. Obviously, each module in the proposed architecture is assigned to the corresponding dequeue and the queuing mechanism takes care of efficient, flexible and easily maintainable data flow through the TTS system. The presented mechanism also enables process interruption after any dequeue in the system, and monitoring evaluation of the corresponding outputs (e.g. output from grapheme-to-phoneme conversion, POS tagging modules, efficiency of the unit selection search algorithm etc.). The HRG structure collects all linguistic information extracted from the input text by the TTS engine modules. The linguistic information from the database sentences stored in the form of HRG structure can also be used for performing resynthesis experiments.

As seen in Figure 9, finite-state machines are used for the separation of language-dependent resources from a language-independent TTS engine. Since finite-state machines are time and space efficient, they are also used for the representation of all language-dependent language resources. Either finite-state automata or finite-state transducers can be used. The FSM compiler must be used for compilation of regular expressions into the finite-state machine, construction of finite-state machine based tokenizers etc. For solving disambiguity problems, heuristically defined or trained weights can be assigned to FSM transitions and final states, yielding weighted finite-state automata and transducers (WFSA, WFST) (Mohri, 1995). The tokenizer is marked as 'T' in the proposed architecture. Additionally, two-level rules or rewrite rules can be used, compiled into finite-state machines by an FSM compiler. These rules can resolve much language-dependent disambiguity in the input texts. Then follows finite-state automaton 'S' for storing large lists of valid words. It can be used by the spell-checking system, when it is included in the architecture. Namely, it is expected that the TTS system will be able to process any input text. The input text often contains more or less spelling mistakes, especially in the case of e-mails or SMS messages. The spell-checking system must be able to detect invalid words and try to guess the most suitable replacements. On the other hand, a POS-tagging module needs large-scale morphology lexicons. The overall performance of the system depends on the time and space efficiency of each module. The finite-state transducer 'P' can be used for time and space efficient representation of large-scale morphology lexicons. Some TTS systems use rule-based POS-tagging algorithms (e.g. Brill, 1993). The obtained POS-tagging rules can be compiled into finite-state machines (Emmanuel and Schabes, 1997). Common POS-tagging processing time only depends on the length of the input sentence and not on the size of the morphological lexicon or the number of rules. The grapheme-to-phoneme conversion module has a significant impact on the final quality of the TTS system, since it defines how to pronounce the input sentence. Advanced systems use large-scale phonetic lexicons for common words, proper names and even foreign words, found in the input text. All such resources can be represented by finite-state transducer (FST) 'G', as seen in Figure 9. Machine-learned models can be used (CARTs, NN etc.) for processing unseen words (words not found in the lexicons). Decision tree models are used in the proposed system, since they represent efficient knowledge representation, regarding time and space requirements. Decision tree models can also be used in the prosody modules (symbolic and acoustic prosody) for the prediction of phrase breaks, prominence and intonation event labels, segment durations, pauses between segments and the acoustic parameters of intonation events. It was shown that decision trees can also be represented by finite-state machines (labeled as WFST 'SP', WFST 'AP' in Figure 9) (Sproat and Riley, 1996; Mohri and Sproat, 1996). However, their compilation into finite-state machines, as in the proposed architecture, only makes sense when they are going to be merged with other finite-state machines, as they are already efficient knowledge representation structures. The unit selection search process represents, in corpus-based TTS systems, a significant time and space issue, because large unit search space exists. Here, finite-state machines can be used for efficient access to unit candidates in the acoustic inventory. Tree-based clustering algorithms can be used for the reduction of large search space, and the dynamic algorithms (e.g. Viterbi algorithm) can be used when searching for such unit sequences that have the best match with the defined prosody for the input sentence. In the concatenation and acoustic modules, digital signal processing algorithms are mostly used for the processing of concatenation points, and for adapting unit candidate pitch and duration. No external language resources are needed in the last two modules.

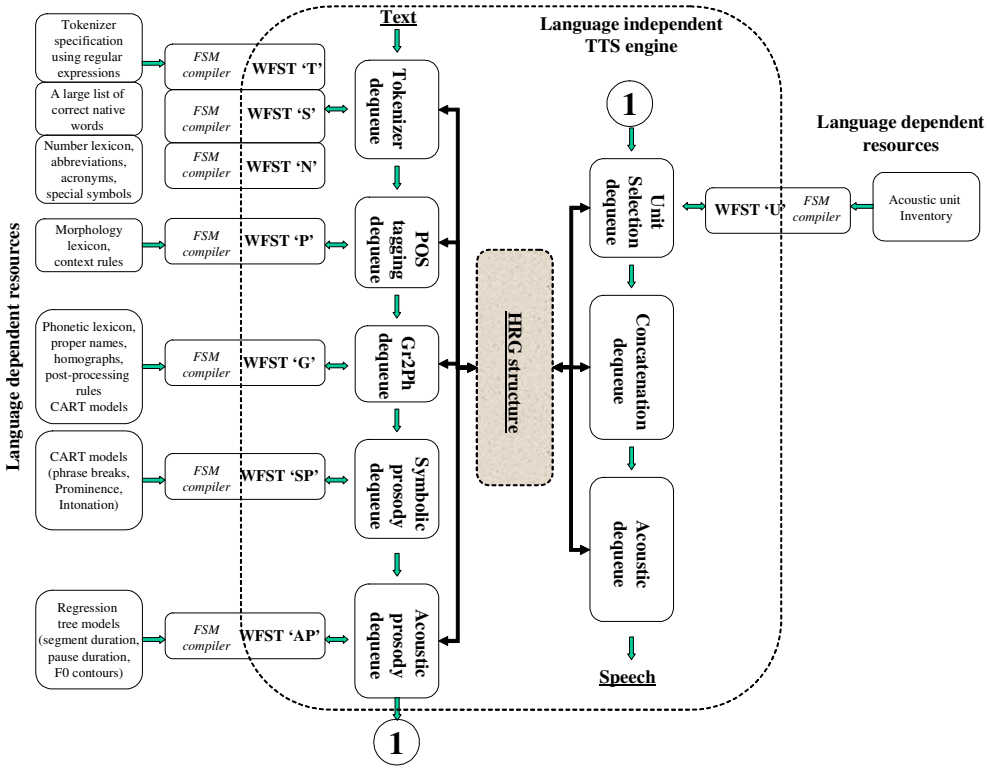


Figure 9: The proposed architecture of the corpus-based TTS system, separated into language-dependent and language-independent parts.

5. Implementation of the proposed architecture within the PLATTOS TTS system

This section presents the implementation of the proposed architecture into the PLATTOS TTS system (Rojc, 2003). The PLATTOS TTS system is a corpus-based speech synthesis system for the Slovenian language, using a concatenative approach and TD-PSOLA speech synthesis algorithm. The dequeues are tied together into a common TTS engine, using the heterogeneous relation graph structure for representation of linguistic information, as shown in the proposed architecture in Figure 9. Finite-state machines, however, are used for language resource representation and separation of the language-dependent part from the language-independent TTS engine. The *fsmHal* library was constructed to efficiently construct the necessary finite-state machines used in the PLATTOS TTS system (Rojc, 2000; Rojc, 2003). In fsmHal library, classical and extended algorithms have been implemented with performance and code usability in mind.

All modules, defined in the general architecture of the TTS system given in Figure 1 are included in the PLATTOS TTS system. In the following subsections, implementation solutions for all modules of the TTS system will be presented in more detail.

5.1. Tokenizer dequeue

All tokens are defined off-line by using regular expressions. The FSM compiler is used for the construction of a tokenizer finite-state machine (Rojc, 2003). The local extension algorithm (Emmanuel and Schabes, 1997) was used, in order for the tokenizer to work globally on the input text. The spell checker in the PLATTOS system is part of the tokenizer module. Usually, people do not read mispelled words aloud but try to correct them (considering the context or even guessing) before pronouncing them. The tokenizer module tries to imitate this habit. It also makes sense, since erroneous words corrupt the performance of all modules in the TTS system, e.g. obtained prosody patterns can result in speech signals with lower intelligibility. The spell-checking algorithm in the PLATTOS system uses a large word list, containing a set of valid words (currently 74,880). Represented as FSA, the corresponding list is efficiently used for edit distance calculations performed when searching for the best possible replacements for the mispelled words found in the input text (Daciuk, 1998; Rojc, 2003). Table 1 shows data of the corresponding finite-state machine, used in the spell-checking system.

Spell checker	FSA
Number of entries	74,880
Number of states	6,638
Number of transitions	15,878
Final size	61 kB

Table 1: FSA representing valid native words, used in the spell-checking system.

Normalisation is also part of the tokenizer module. Quite a lot of tokens in texts are not found in word forms (numbers, special symbols - $, %, acronyms etc.). Factorization is performed firstly in order to convert numbers into the corresponding word forms. Some languages (e.g. German and Slovene) have well-known phenomena named decade flop, so an additional filter is used in the case of the Slovenian language, for handling such language specific phenomenon (Sproat, 1998; Rojc, 2003). This FST is language-specific. Then number lexicon constructed from SIlex lexicon (Rojc and Kačič, 2000) is represented as FST. Furthermore, rewrite rules are used for language specific word insertions (special words such as "and" (English), "und" (German) or "in/and" (Slovene). Compiling rewrite rules into a FST can be achieved using two algorithms. One was proposed in Kaplan and Kay, 1994,and the other in Mohri and Sproat, 1996. The latter is used in the Plattos TTS system, since it is more efficient and requires a limited number of operations (Arnaud, 2000). Table 2 presents those finite-state transducers representing the cardinal and ordinal numbers' lexicons. Very small machines are also obtained for the factorisation FST and decade-flop FST, as shown in Table 3.

Cardinal number lexicon	FST	Ordinal number lexicon	FST
Number of entries	3,454	Number of entries	4,585
Number of states	112	Number of states	64
Number of transitions	679	Number of transitions	512
Final size	12 kB	Final size	8 kB

Table 2: Cardinal number lexicon (FST) and ordinal number lexicon (FST) obtained from SIlex lexicons.

Factorisation	FST	Decade flop	FST
Number of states	313	Number of states	249
Number of transitions	433	Number of transitions	492
Final size	8 kB	Final size	9 kB

Table 3: Factorisation (FST) and decade-flop transducers (FST).

The normalisation process of abbreviations is an important issue, especially for inflectional languages such as Slovenian. Firstly, the construction of an FST is performed, which contains all possible word forms for a given abbreviation (e.g. kg). This is carried out by an expert, who writes down a list of corresponding regular expressions. These regular expressions are compiled automatically into a finite-state machine using a regular expression compiler. Finally, a decision has to be made as to which conversions are possible and which are impossible, when considering the context. The marking of acceptable and unacceptable conversions for a given context is done using rewrite rules, written by an expert. After the conversions are marked as acceptable or unacceptable, the latter are removed using a filter transducer (Sproat, 1998; Rojc, 2003).

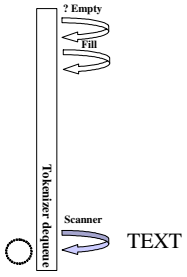


Figure 10: Tokenization dequeue.

When processing special symbols (e.g. %), the construction of FSM representing lexical analysis for a given symbol, is firstly performed for the conversion of a special symbol into word forms. Here, even at the beginning, all conversions are marked as unacceptable and some of them are removed during the filtering operation, in case there is no context defined that should preserve them. Defining corresponding contexts is again performed using weighted rewrite rules compiled

into weighted finite-state transducers (WFST). In the cases where more possible conversions are preserved at the end, the most appropriate one is obtained using *BestPath* algorithm (Sproat, 1998; Rojc, 2003).

In the queuing mechanism, the tagger dequeue sends demands for tokens to the tokenizer dequeue in Figure 10, until all sentence tokens are accepted. In order for the architecture to know when the 'end of sentence' condition is met, the tokenizer module inserts an additional token, named T_EOS, into the dequeue. An additional token for 'end of file' condition or "no more text" condition, is named T_EOF and is also inserted by the tokenizer module. No items are added to the HRG utterance structure by this module. Tokens extracted from the input text are simply moved to the tagger dequeue, using the *pushtoken()* function defined in the abstract class named *TTSEngine*. The following result can be, for example, observed after the tokenizer module:

Tokenizer> dvesto/T_WORD deset/T_WORD centimetrov/T_WORD ./T_EOS (two hundred centimetres)

5.2. POS tagging dequeue

The POS tagging approach is performed in the PLATTOS TTS system that is similar to Brill's POS tagging approach (Brill, 1993). Brill's unsupervised transformation-based error driven training algorithm is used, where the expert follows the following procedure:

- manually marks a small part of the untagged corpus
- performs training on this corpus
- automatically tags new sentences
- removes mistakes and again performs training
- tags new sentences

The POS tagging process consists of more steps. Firstly, the morphology lexicon obtained from the training is used. In this lexicon each entry is assigned the most probable POS tag found in the training corpus. If a word is not found, the SImlex morphology lexicon is used (Rojc and Kačič, 2000). Deterministic and minimized FST representation of the lexicon gives time and space-efficient representation and fast lookup time. Input and output strings for each entry, stored in the FST, are not necessarily of equal length. Therefore, filler symbols are used for filling up the shorter string. Reduction of the FST size is improved by inserting filler symbols in appropriate positions, since the number of equal transitions is increased. Good positions are those positions that align appropriate segments in both strings. Such representation gives smaller machines, since word beginnings consist mostly of pairs of identical characters, and endings have the same mapping for the suffixes of the input strings into output strings. Next comes morphological analysis, which uses the so-called guessing automata, constructed for unknown words (FSA that tries to guess the POS tag by analysing word endings (Daciuk, 1998)). Finally, POS tagging context rules are used. In the post-processing stage, local grammars are used to resolve remaining ambiguities, as a consequence of systematic tagging errors that are unsolved during the POS-tagging process (Brill, 1993; Roche and Schabes, 1995; Rojc, 2003). Table 4 shows data for the finite-state transducers regarding morphological lexicons. Algorithm for the morphology analysis of unknown words needs a list of words with assigned morphological information. As shown in Table 5, it can be efficiently used when represented as finite-state automaton. Context rules, defined from annotated training corpora, can also be represented very efficiently by FST (Roche and Schabes, 1995; Rojc, 2003).

SImlex morphology lexicon	**FST**	**POS-tagging lexicon**	**FST**
Number of entries	81,208	**Number of entries**	17,200
Number of states	193,559	**Number of states**	32,277
Number of transitions	244,714	**Number of transitions**	45,926
Final size	911 kB	**Final size**	193 kB

Table 4: POS-tagging lexicons represented as FST.

Guesser (for unknown words)	**FSA**	**Brill's context rules**	**FST**
Number of entries	113,913	**Number of rules**	331
Number of states	188,305	**Number of states**	1,720
Number of transitions	239,651	**Number of transitions**	22,076
Final size	733 kB	**Final size**	365 kB

Table 5: POS-tagging resources for unknown words represented as FST.

In the queuing mechanism the tagger dequeue sends demands for tokens to the tokenizer dequeue, until T_EOS token is accepted. POS tagged word items are inserted into *Word* linear relation structure defined in the HRG utterance structure, as seen in Figure 11. To each item word POS tag attribute is assigned.

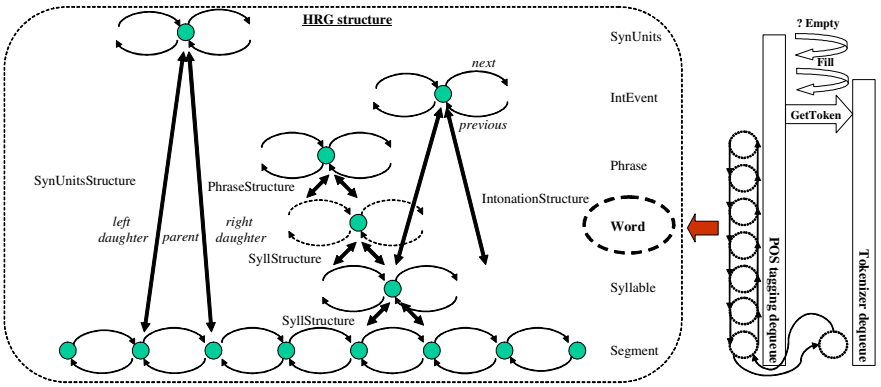


Figure 11: POS tagging dequeue.

After POS tagging, the tokens in the dequeue are demanded by the grapheme-to-phoneme conversion module. The following result can be, for example, observed after POS tagging module, where Multext-East POS tags are used (Rojc, 2003):

Tagger> dvesto/Mcmpnl deset/Mcfpnl centimetrov/Ncmpg ./T_EOS

5.3. Grapheme-to-phoneme conversion dequeue

The SIflex phonetic lexicon (Rojc and Kačič, 2000) for common words is first used in the grapheme-to-phoneme conversion (G2P). Then, the SIplex phonetic lexicon for proper names is used (Kačič, 1995), followed by the homograph detection step. All unknown words are converted into phonetic transcription using CART tree models and, finally, the set of post-processing rules are used to adapt phonetic transcriptions regarding cross-word contexts. All lexica are represented as FST. Table 6 shows finite-state transducers for SIflex (common words) and SIplex (proper names) phonetic lexicons.

SIflex	**FST**	**SIplex**	**FST**
Number of entries	68,817	**Number of entries**	237,657
Number of states	23,301	**Number of states**	717,867
Number of transitions	43,006	**Number of transitions**	926,014
Final size	197 kB	**Final size**	4 MB

Table 6: Phonetic lexicons SIflex (common words) and SIplex (proper names), represented as FST.

A very important issue is the construction of those machine-learned models used for the grapheme-to-phoneme conversion of unknown words. Classification trees are used for this task (CART trees), (Breiman et al., 1984). Firstly, alignment of orthographic and phonetic strings is performed, by using the epsilon scattering method, where the probabilities of mapping grapheme L into phoneme P are estimated and the dynamic time warping (DTW) method is used to find the best cost effective alignments, inserting *epsilon* symbols where appropriate (Pagel et al., 1998). Heterogeneous relation graphs are used for the representation of linguistic knowledge for phonemes and corresponding words. Various complex features can be efficiently constructed by using a simple textual list of the linguistically attributed names used in the HRG structure. After grapheme-to-phoneme conversion, syllable markers have to be inserted in case of unknown words, since this information is very important for prosody modules in the next stages of the TTS system (Kiraz and Möbius, 1998; Rojc, 2003). The classification tree model is also used for the insertion of syllable marks.

In the final stage of G2P, post-processing rules are used that perform the post-processing of the canonic phonetic transcriptions, by considering cross-word contexts. All phonetic transcriptions are namely determined without considering any cross-word context within the sentence. It is well-known in the Slovenian language that cross-word context has a significant impact on pronunciation and must be considered within the whole grapheme-to-phoneme conversion process. The expert defines the rules of these phoneme conversions, occuring at word beginnings and word endings. The obtained rules can also be represented by FST (Sproat, 1998).

The off-line trained CART models for stress, grapheme-to-phoneme, and syllable prediction are used in the on-line G2P process for unknown words when found in the input text. Table 7 shows the sizes of the stress prediction, grapheme-to-phoneme conversion, and syllable prediction models.

Stress prediction model	Gr2Ph prediction model	Syllable prediction model
354 kB	768 kB	160 kB

Table 7: CART models used in the grapheme-to-phoneme module for unknown words.

Often texts contain words or phrases in some other language (secondary language, or even ternary language). The first problem is to detect such words, and the second is to define the corresponding pronunciations. Use of phonemes from a secondary language is, in our case, not an option. The recording of databases with the same speaker in different languages is an unacceptable solution, since the more languages the TTS system includes the harder it is to find a speaker who is able to speak all these languages fluently. The procedure proposed in Rojc, 2003 is implemented in the PLATTOS TTS system. For example, if we take the input sentence in the Slovenian language containing the name "Gerhard Schröder", which is a German name within a Slovenian sentence. These words are detected and converted into the corresponding phonetic transcriptions using a grapheme-2-phoneme conversion module for the German language (using SIplex lexicon). The obtained sequence of German phonemes is then mapped into the most suitable corresponding substitutions found among the Slovenian phonemes. This mapping must be done by using the table constructed by the phonetic expert. Table 8 shows the finite-state automata used for the detection of foreign words (German and English), where automata were constructed from Süd Deutsche Zeitung (German) and Reuters Corpus (English).

Detection of foreign words	Süd Deutsche Zeitung FSA	Reuters Corpus FSA
Number of entries	89,885	44,450
Number of states	79,688	41,209
Number of transitions	121,321	87,098
Final size	579 kB	253 kB

Table 8: FSAs used for polyglot grapheme-to-phoneme conversion process.

The complete G2P process shown in Fig. 8-1 is represented as a sequence of finite-state transducers that can also be merged into common FST, using composition operation (Mohri, 1997).

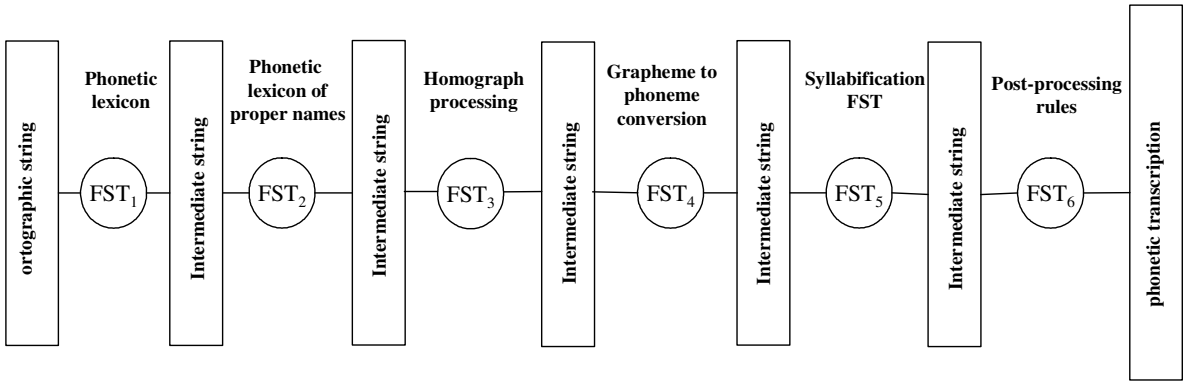


Figure 8-1: The cascade structure of FSTs for grapheme-to-phoneme conversion.

In the queuing mechanism, the grapheme-to-phoneme conversion dequeue sends demands for tokens to the POS tagging dequeue, until T_EOS token is accepted. After grapheme-to-phoneme conversion of all word items in the HRG structure, segment items and syllable items are defined and added to the HRG utterance structure. As previously mentioned, the *Word*, *Syllable* and *Segment* relation structures are linear lists. In order to obtain more complex linguistic features,

additional vertical tree relation structures are established between these linear lists, named as *SyllStructure*. Syllable and segment items' attributes as: end of syllable, end of word, phoneme type, stress type, stress position etc., are also assigned.

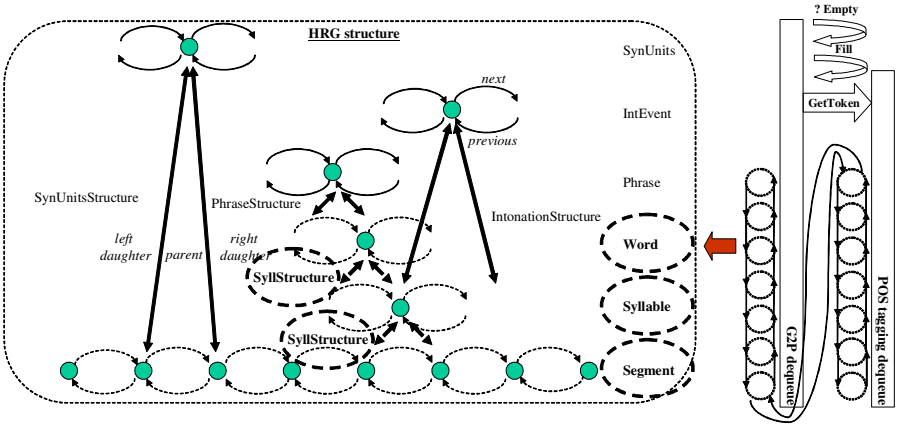


Figure 12: Grapheme-to-phoneme conversion dequeue.

After grapheme-to-phoneme conversion, the tokens in the dequeue are demanded by the symbolic prosody module. The following result can be observed after grapheme-to-phoneme conversion, where SAMPA symbols are used:

Gr2Ph> dvesto/d v /e: – s t O deset/d E – s /e: t centimetrov/ts E n – t i – m /e: – t r o U ./T_EOS

5.4. Symbolic prosody dequeue

The symbolic prosody module performs three tasks: prediction of phrase breaks, prediction of prominence labels, and the prediction of Tilt intonation labels on syllables (Strom, 1998; Taylor, 2000; Rojc, 2003). Classification trees (CART trees) are used in the symbolic prosody module, since we perform classification of discrete linguistic values. The phrase break prediction model inserts phrase break labels in the input text, the prominence prediction model marks the prominent syllables, and the intonation prediction model assigns Tilt intonation labels to each syllable (Taylor, 2000). The purpose of prosodic phrase breaks is to achieve more natural speech synthesis. Usually, phrase breaks are set at punctuation positions inside the sentence, but can also be present at positions where there are no punctuation symbols. In the phrase break prediction system, a B3 label is used for labelling major phrase breaks and B2 label for minor phrase breaks. Phrase break positions are also used for pause insertions in the sentence.

Prominence labels on syllables are predicted by CART trees and are marked as PA (primary accent that is assigned to the most accentuated syllables inside the intonation prosodic phrase - labelled with B3) and as NA (marking secondary accents in the prosodic phrase).

CART trees are used for the prediction of Tilt intonation labels. The Tilt intonation event labels are assigned to each syllable in the sentence. The following Tilt intonation labels for corresponding intonation events are used in the system: *a c l m fb rb afb arb lfb mrb mfb lrb* (Taylor, 2000; Rojc, 2003).

Table 9 shows the sizes of CART models for phrase break prediction, prominence prediction and Tilt intonation prediction models.

Phrase break prediction model	Prominence prediction model	Intonation prediction model
474 kB	218 kB	900 kB

Table 9: CART models used in symbolic prosody module.

In the queuing mechanism, the symbolic prosody dequeue sends demands for tokens to the grapheme-to-phoneme conversion dequeue, until T_EOS token is accepted. CART models are used for prediction of phrase breaks, prominence labels and Tilt intonation event labels on the current utterance. During symbolic prosody processing, phrase break items are added into *Phrase* relation structure. Prominence labels are assigned as additional attributes to syllable items in the *Syllable* relation structure. Tilt intonation events are added as new items into the *IntEvent* relation structure. In order to

obtain more complex linguistic features, additional vertical tree relation structures are established between *Phrase* and *Word* linear relation structures, named as *PhraseStructure*, and between *IntEvent* and *Syllable* linear relation structures, named as *IntonationStructure* as seen in Figure 13.

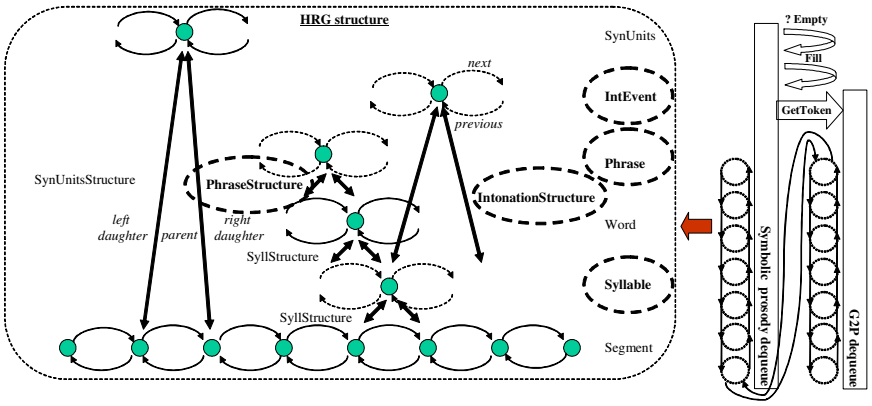


Figure 13: Symbolic prosody dequeue.

After symbolic prosody module, the tokens in the dequeue are already demanded by the acoustic prosody module. The following result can be, for example, observed after symbolic prosody step:

SymProsPB> dvesto/(B2) deset/(*) centimetrov/(B3) ./T_EOS

SymProsAcc> dvesto/d v /e: (NA) s t O (*) deset/d E (*) s /e: t(NA) centimetrov/ts E n (*) t i (*) m /e: (PA) t r o U (*) ./T_EOS

SymProsInt> dvesto/d v /e: (a) s t O (c) (sil) deset/d E (c) s /e: t (a) (sil) centimetrov/ts E n (c) t i (*) m /e: (c) t r o U (m) (sil) ./T_EOS

5.5. Acoustic prosody dequeue

The acoustic prosody module performs three tasks: prediction of segment durations, prediction of pause durations at phrase break positions and prediction of Tilt acoustic parameters for corresponding Tilt intonation events assigned to syllables in the symbolic prosody module (Taylor, 2000; Rojc, 2003). Regression trees are used as machine-learning models, because of the nature of the acoustic data (continuous values), (Breiman et al., 1984).
Two separate models are defined for the segment duration prediction model: one for the prediction of vowel phoneme durations and the other for the prediction of consonant phoneme durations. The regression tree model for the prediction of pause durations is trained using only those internal pauses found in the PLATTOS speech database. The pauses at the start and at the end of the database sentences were not included, since they had very different and often unnatural lengths. The PLATTOS speech database contains neutral read speech, therefore, the assumption that the internal pauses have, in most cases, natural length seems to be reasonable. In the symbolic prosody module, Tilt intonation labels were assigned to each syllable. The second step represents the prediction of corresponding Tilt acoustic parameters for each Tilt intonation event. When Tilt acoustic parameters are defined by the regression tree models, reconstruction of the F0 contour can be performed. The regression tree models are an efficient representation of the acoustic knowledge (segment and pause durations, Tilt acoustic parameters) obtained during the processing of the PLATTOS speech database (Rojc, 2003). Table 10 shows the sizes of the regression tree models for duration prediction (vowels, consonants, pauses) and Tilt acoustic parameters prediction.

Vowel duration model	Consonant duration model	Pause duration model	Tilt intonation models
2.094 MB	635 kB	527 kB	3.36 MB

Table 10: Regression tree models used in the acoustic prosody module.

In the queuing mechanism, the acoustic prosody dequeue sends commands for tokens to the symbolic prosody dequeue, until T_EOS token is accepted. Regression tree models are used for the prediction of segment durations, pause durations and Tilt acoustic parameters on the current utterance, based on the linguistic information stored in the common HRG utterance structure. During acoustic prosody processing, segment and pause durations are assigned as additional attributes to existing items in the *Segment* relation structure. Tilt acoustic parameters are added as additional attributes to items in

the *IntEvent* relation structure, as shown in Figure 14. After acoustic prosody module, the tokens in the dequeue are demanded by the unit selection module.

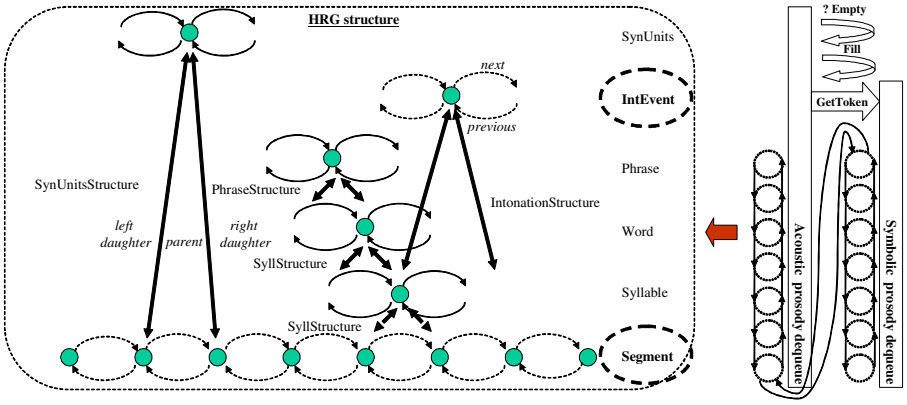


Figure 14: Acoustic prosody dequeue.

5.6. Unit selection dequeue

The unit selection module also uses external language dependent resources. They can mostly be prepared off-line during the processing of the PLATTOS speech database. Automatic and semi-automatic methods and algorithms are used, in order to minimize the needed manual intervention of the expert (Rojc, 2003).

The unit selection model uses acoustic inventory, consisting of units that are assigned with acoustic information (duration, pitch marks, sample position etc.), (Black and Campbell, 1995; Black and Taylor, 1997). The PLATTOS TTS system is a corpus-based system that uses diphones and triphones as basic acoustic units. In the optimisation step, acoustically too similar units are removed from the acoustic inventory. The acoustic measures used are energy, duration, and pitch. The implemented optimisation algorithm is based on fuzzy logic formalism (Holzapfel, 2000). In order to minimize the search space, the unit selection algorithm uses a tree-based clustering procedure that classifies units into clusters, regarding the phonetic context (Rojc, 2003).

The concatenation costs between all units in the acoustic inventory are calculated off-line. The concatenation cost matrix is, in the case of corpus-based synthesis, very large (e.g. for 300,000 units). In order to obtain efficient representation of the matrix, vector quantisation based compression is used and indices in the constructed codebook are represented by a finite-state machine (Rojc, 2003).

In the on-line TTS system, the unit selection dequeue loads the acoustic inventory, trees with clusters of diphone and triphone units, a vector quantisation table representing concatenation costs between units, and the FST transducer with indices, into the vector quantisation table. The unit selection algorithm first finds clusters of candidates for each sentence target unit. In the second step it performs Viterbi search, where the vector quantisation table and FST with indices into the vector quantisation table are used for the assignment of concatenation costs to network transitions between units. Table 11 shows the size of the concatenation cost matrix, as defined by the number of diphone units $N_{diphones}$ and by the number of triphone units $N_{triphones}$. The corresponding weights are represented in the form of a vector quantisation table (CC_{costs}). The corresponding indices into the vector quantisation table are represented as FST ($FST_{indices}$). The last column represents the size of the speech database (16 kHz, raw format).

$N_{diphones}$	$N_{triphones}$	CC_{costs}	$FST_{indices}$	Speech samples
172,588	165,261	133 kB	2,53 MB	276 MB

Table 11: Resources used in unit-selection dequeue for 337,849 units.

The total number of all diphones and triphones in the acoustic unit inventory after optimisation (removing acoustically too similar units) is 136,849 (Rojc, 2003). The complete database consists of a total of 337,849 diphones and triphones. The speech samples are not loaded directly into the memory. They are 16 kHz/16 bit samples stored in the raw format. The speech database is only opened at the start of the TTS processing. The speech samples of corresponding units are only loaded after the unit selection process, when the list of best candidates has already been found.

In the queuing mechanism, the unit selection dequeue sends demands for tokens to the acoustic prosody dequeue, until T_EOS token is accepted. The unit selection module adds items into the linear list relation structure named *SynUnits*. The inserted items represent all those unit candidates selected from the acoustic inventory by unit selection algorithm. An additional tree relation structure named *SynUnitsStructure* is established in order to link *SynUnits* and *Segment* relation structures. This vertical link and corresponding tree relation structure is needed as items in *Segment* relation structure that contain information about segment and pause durations are defined by the acoustic prosody module (Figure 15). Finally, at the end of the unit selection process, the relation *SynUnits* contains a sequence of unit candidates that have the best match with the predicted prosody, with corresponding samples. After unit selection module, the tokens in the dequeue are commanded by the concatenation module.

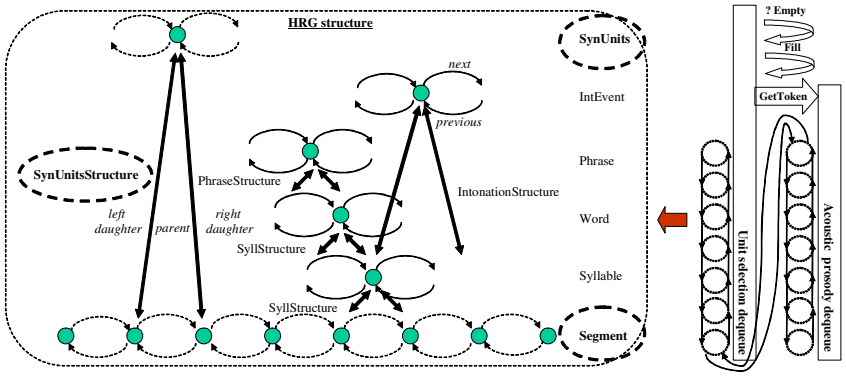


Figure 15: Unit selection dequeue.

5.7. Concatenation dequeue

Concatenation module processes concatenation points between those units selected by the unit selection process. In this module the following processing steps are performed: calculation of analysis pitches, searching for an optimal concatenation point between two successive units, matching of analysis and synthesis pitches and the smoothing of concatenation points. No external language-dependent resources are needed (Rojc, 2003).

In the queuing mechanism, the concatenation dequeue sends demands for tokens to the unit selection dequeue, until T_EOS token is accepted. At this level, no additional items or attributes are added to the HRG structure, only items in the *SynUnits* relation structure are used during processing (Figure 16). After the concatenation module, the tokens in the dequeue are demanded by the acoustic module.

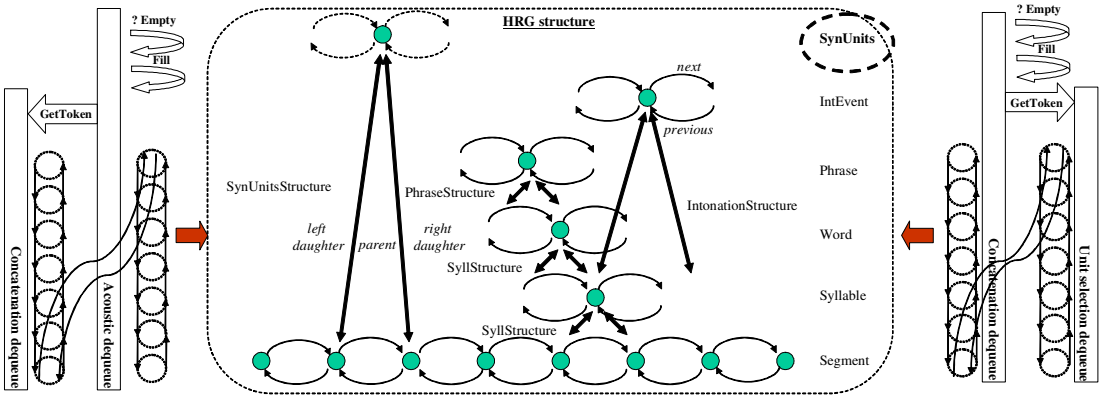


Figure 16: Acoustic and concatenation dequeue.

5.8. Acoustic dequeue

The acoustic module performs TD-PSOLA algorithm used for changing the duration and pitch on those selected units stored in the HRG structure. This algorithm is used in order to carry out an adaptation of the unit acoustic prosodic

parameters (duration and pitch) according to the desired prosody of the input sentence. The better the match between the selected units and desired prosody, the less additional signal processing is needed (Rojc, 2003).

In the queuing mechanism, the acoustic dequeue sends demands for tokens to the concatenation dequeue, until T_EOS token is accepted. Also at this level, no additional items or attributes are added to the HRG structure, only items in the *SynUnits* relation structure are used during processing (Figure 16). The acoustic module does not need any external language resources and does not use any finite-state machines for its operation. After acoustic module, the main process of the queuing mechanism detects the end of sentence condition and is, therefore, able to release the memory used by the HRG structure and tokens. If the condition 'no more text' is not met, the queuing mechanism starts to process the next sentence.

5.9. Overall structure of the TTS system PLATTOS

Figure 17 shows the overall structure of the PLATTOS corpus-based TTS system, based on the proposed architecture. The dashed-line large rectangle, denotes the language-independent TTS engine. The language-dependent resources are represented by using finite-state machine formalism, and CART models. Constructed FSMs and CART models are loaded into a TTS engine in a uniform way. These language-dependent models are constructed off-line by using PLATTOS tools (Rojc, 2003). As can be seen from Figure 17, the following language resources are needed: regular expressions for tokenizer construction and text normalisation, large list of valid native words, number lexicon, acronym lexicon and lexicon of special symbols, morphology lexicon, phonetic lexicons, homograph database and phonetic post-processing rules, prosodically annotated speech database (phrase breaks, prominence, intonation) used for training prosodic CART models, and acoustic inventory constructed from the speech database. All this data has to be available for the target language in order to achieve the maximum performance of the whole system. For the Slovenian language, the language-dependent resources used are marked as dashed-line rectangles as in Figure 17: Slovenian newspapers, SIlex lexicons and PLATTOS speech database.

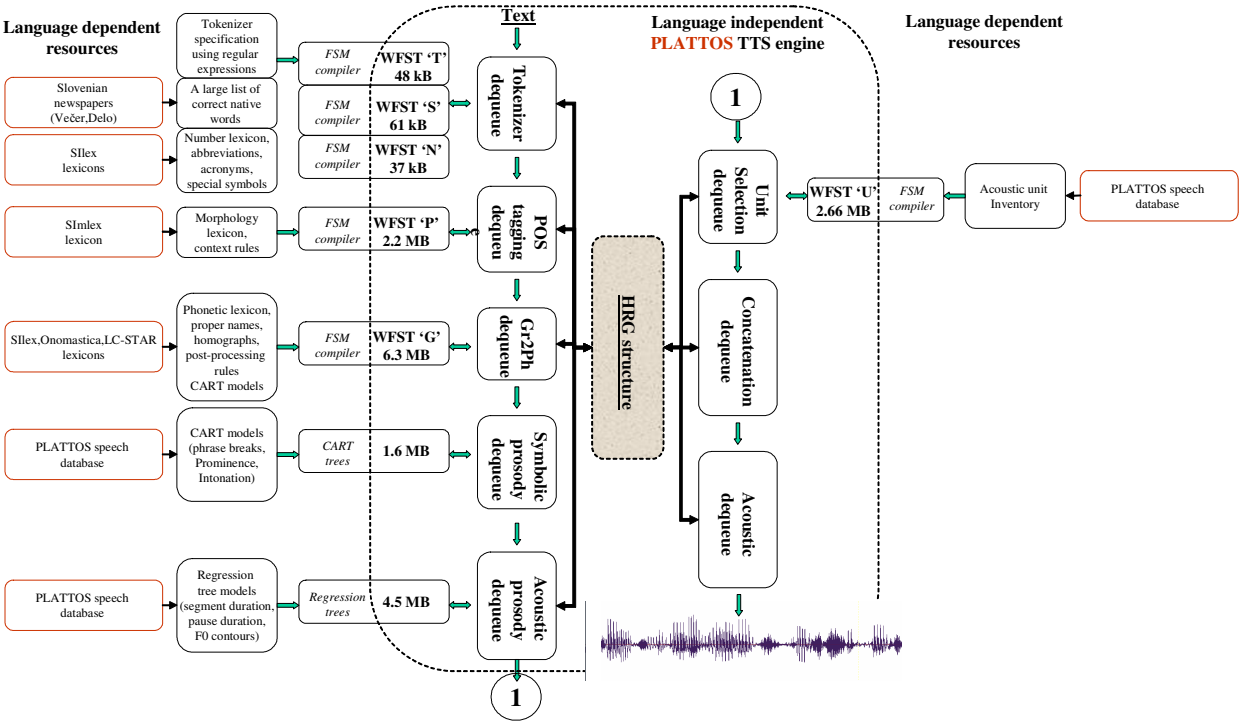


Figure 17: The TTS system PLATTOS, with separated language-dependent and language-independent part.

In the PLATTOS TTS system, the tokenizer module consists of tokenization FSM, spell-checking FSM and normalisation FSM. Altogether this represents 146 kB. The POS tagging module, which uses morphology lexicons and POS tagging context rules, requires altogether 2.2 MB. Grapheme-to-phoneme conversion module uses phonetic lexicons and CART

models. All resources together use 6.3 MB. Symbolic prosody module uses CART trees that represent 1.6 MB. The acoustic prosody module uses regression trees that represents 4.5 MB. The unit selection module uses the concatenation costs codebook and corresponding FST with indices. These resources represent altogether 2.66 MB. Samples are not loaded directly into the memory and are loaded, on demand, from the hard disc. Therefore, all language dependent resources together amount to 17.3 MB.

By using SIlex lexicons in the PLATTOS system, coverage of up to 85% is achieved on general texts. The developed architecture benefits from data structures such as finite-state machines and heterogeneous relations graphs, is easily maintainable, allowing flexible migration to new languages, has efficient data flow throughout the whole system and between modules, and allows easy monitoring and performance evaluation after each module. The current level of optimisation performed in all modules of the system and representation of language resources doesn't affect the final quality of the synthesised speech. When the speech quality can be degraded, further optimisation on the system is still possible, resulting in a smaller footprint of the corpus-based TTS system.

6. Conclusion

This paper presents time and space-efficient architecture for a text-to-speech synthesis system. It shows that it is possible to integrate all parts of the TTS system, from text processing to acoustic processing, into an efficient and flexible queuing mechanism. All modules can use time and space-efficient finite-state machines for separating language-dependent resources from a language-independent TTS engine, for time and space efficient representation of language resources and for fast information lookup. A heterogeneous relation graph can be used for storing very heterogeneous linguistic information flexibly and efficiently. It can also be used for flexible construction of complex features. Using the proposed architecture, only language-dependent resources have to be prepared for the development of a text-to-speech synthesis system for a new language. The corresponding FSM compilers (Rojc, 2003) must be used for the representation of linguistic knowledge by finite-state machines, and for their integration into the TTS system. The architecture is uniform, since it does not use different structures and machine-learned models. The proposed architecture is implemented in the Slovenian corpus-based PLATTOS TTS system, however, it can be used for the construction of TTS systems for any language, for which the necessary language resources exist. Performance of corpus-based TTS systems mainly depends on the size of the acoustic unit inventory and on the size of the unit search space. The proposed TTS architecture is very flexible and many different configurations of the system are possible, each having specific 'speech quality' / 'realtime factor' ratio. Current implementation of the text-to-speech synthesis system converts text into speech in real time on Intel P4 2.53 GHz (using the complete database).

7. References

Aho, Alfred V., John E. Hopcroft, and Jeffrey D. Ullman. 1974. *The design and analysis of computer algorithms*. Addison Wesley: Reading, MA.

Arnaud, Adant. 2000. *Study and implementation of a weighted finite-state library – application to speech synthesis*. Faculté Polytechnique de Mons, Belgium.

Black, Alan. W., Campbell N. 1995. *Optimising selection of units from speech databases for concatenative synthesis*. In Eurospeech95, volume 1, pages 581-584, Madrid, Spain.

Black, A. W. and Taylor, P. 1997. *Automatically Clustering Similar Units for Units Selection in Speech Synthesis*. Proceedings of Eurospeech, 2:601-604.

Breiman, L., Freidman, J., Olshen, R. and Stone, C. 1984. *Classification and Regression Trees*. Chapman & Hall, New York.

Brill, E. 1993. *A Corpus-Based Approach to Language Learning*. PhD thesis.

Bulyko, I. 2001. *Unit Selection for Speech Synthesis Using Splicing Cost with Weighted Finite State Transducers*. In Proc. of Eurospeech.

Bulyko, I., and Ostendorf, M. 2001. *Joint prosody prediction and unit selection for concatenative speech synthesis*, In Proc. of ICASSP.

Campbell, N., A. Black. 1996. *CHATR: a multi-lingual speech re-sequencing synthesis system*. Institute of Electronic, Information and Communication Engineers, Spring Meeting, Tokyo SP-96-07.

Clark, Robert A.J., Richmond K., and King, S. 2004. *Festival 2 – build your own general purpose unit selection speech synthesiser*, In Proc. 5th ISCA Workshop on speech synthesis.

Daciuk, J. 1998. *Incremental Construction of Finite-State Automata and Transducers and their Use in the Natural Language Processing*, Ph.D. thesis, Technical University of Gdansk, Poland.

Emmanuel, R., Schabes Y. 1995. *Deterministic Part-Of-Speech Tagging with Finite-State Transducers*. Mitsubishi Electric Research Laboratories.

Emmanuel, R. and Schabes Y. 1997. *Finite State Language Processing*. The Massachusetts Institute of Technology.

Holzapfel, M. 2000. *Konkatenative Sprachsynthese mit grossen Datenbanken*. Ph.D. thesis.

Horowitz, E., Sahni S. 1996. *Computer Algorithms C++*. Computer Science Press.

Kačič, Z. 1995. *Onomastica for Slovenian*. [http://www.elda.fr/catalogue/speech/S0043.html].

Kaplan, Ronald M. and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3).

Karttunen, Lauri, Ronald M. Kaplan, and Annie Zaenen. 1992. Two-level morphology with composition. In *Proceedings of the _fteenth International Conference on Computational Linguistics (COLING'92), Nantes, France*. COLING.

Kiraz, George A., Möbius B. 1998. *Multilingual syllabification using weighted finite-state transducers*. Proceedings of the third International Workshop on Speech Synthesis, Australia.

Mohri, M. 1995. *On Some Applications of Finite-State Automata Theory to Natural Language Processing*. Natural Language Engineering 1.

Mehryar Mohri. 1997. *Finite-state transducers in language and speech processing*. Computational Linguistics, 23(2):269--311.

Mohri, Mehryar, Fernando C. N. Pereira, and Michael Riley. 1996. Weighted automata in text and speech processing. In *ECAI-96 Workshop, Budapest, Hungary*. ECAI.

Mohri, M., and Sproat R. 1996. *An efficient compiler for weighted rewrite rules*. In 34-th Meeting of the Association for Computational Linguistics (ACL 96), Santa Cruz, California.

Pagel, V., Lenzo K. and Black Alan W. 1998. *Letter to Sound Rules for Accented Lexicon Compression*, In Proc. of ICSLP.

Rojc, M. 2000. *The use of finite-state machines for Text-to-Speech systems*. Master thesis.

Rojc, M. 2003. *Time and Space Efficient Architecture of the Multilingual and Polyglot Text-to-Speech System – Architecture with Finite-State Machines*. PhD Thesis.

Rojc, M., Kačič Z. 2000. *A Computational Platform for Development of Morphologic and phonetic lexica*. Proceedings of the Second Language Resources and Evaluation Conference (LREC), Athens, Greece.

Silberztein, Max. 1993. *Dictionnaires ´electroniques et analyse automatique de textes: le syst_eme INTEX*. Masson: Paris, France.

Sproat, R. 1998. *Multilingual Text-to-Speech Synthesis*. Kluwer Academic Publishers.

Sproat, R. and Riley, M. 1996. *Compilation of Weighted Finite-State Transducers from Decision Trees*. In Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics, pp. 215-222.

Strom, V. 1998. *Automatische Erkennung von Satzmodus, Akzentuierung und Phrasengrenzen in einem sprachverstehendem System*. Ph.D. Thesis, Bonn.

Syrdal, A.K., Wightman, C.W., Conkie, A., Stylianou, Y., Beutnagel, M., Schroeter, J., Strom, V., Lee, K-S., and Makashay, M.J. Oct. 2000. *Corpus-based techniques in the AT&T NextGen synthesis system*. Proc. ICSLP, Vol. 3, pp. 410-415, Beijing, China.

Taylor, P., Black Alan W. and Caley R. 1998. *The architecture of the Festival speech synthesis system*, Proc. The Third ESCA Workshop in Speech Synthesis, 1998, pp. 147-151.

Taylor, P. 2000. *Analysis and Synthesis of Intonation using the Tilt Model*. Jorunal of the Acoustical Society of America.

Taylor, P., Black Alan W. and Caley R. 2001. *Heterogeneous relation graphs as a formalism for representing linguistic information*, Speech Communication, Vol. 33 (1-2) 2001 pp. 153-174.

Watson, B.W. 1995. *Taxonomies and Toolkits of Regular Language Algorithms*, PhD thesis, Eindhoven University of Technology.