A Flexible Multilingual TTS Development and Speech Research Tool

Géza Kiss, Géza Németh, *Gábor Olaszy, Géza Gordos
Dept. of Telecommunications & Telematics, Budapest University of Technology and Economics,
*Kempelen Farkas Speech Research Laboratory, Hungarian Academy of Sciences, Hungary
{kgeza, nemeth, olaszy, gordos}@ttt.bme.hu

Abstract
Diverse synthesis methods and text-to-speech (TTS) architectures are being developed and applied almost every day. This tendency raises the need for durable program systems that effectively assist research and development in this area. A flexible development system for multilingual text-to-speech and general speech research is introduced. The system was developed for use with the Multivox and Profivox concatenative speech synthesis systems, but its architecture makes it theoretically appropriate for a wide variety of purposes and different TTS systems. The system architecture and the functions of the development system are described. Keywords: TTS development system, speech research tools, system architecture, SGML derivative, object oriented design

1. Introduction
During the process of developing a TTS system, numerous tasks are accomplished, including the scientific analysis of written and spoken text, rule based creation of TTS components, the creation of databases, design and implementation of prosodic algorithms and test of the completed system. Practically all of these can be aided effectively by some specialized programs. Some tasks can be completely or partly automated and programs can give interactive help to the researcher.

Multiple integrated systems, called TTS development systems, have been created that aim at helping a number of the tasks mentioned for a specific TTS system. The Delta System [1] is designed to help modular creation of TTS systems and give support to linguists when designing the analysis modules, but does not give tools for database creation. The Festival development system [2] assists nearly all phases of TTS system creation: newer versions also aid creating speech databases of a special architecture.

The purpose of our system is twofold. First, creating of TTS development environments has a history of more than 15 years [3]. The present Profivox TTS system [4] is the newest version in this historical background, following its main lines of thought. On the other hand, the system had to support education and general speech research work as well, in a non-engineering environment. Following the development of different computer platforms, the present system can run on the Windows platform, but is open to other ones (e.g. Linux). It had to facilitate the creation process of waveform speech unit databases. It was also important that it should give access in a transparent way to data inside the TTS system without requiring detailed involvement in the software development process itself. A modular configuration of individual subsystems was also required. The paper describes PDS (Profivox Development System), developed in our laboratory for these objectives. PDS is a flexible tool for giving aid to the researcher in experimentation (analysis by synthesis), for the purpose of speech database creation, and for supporting the testing of the internal modules of the TTS system.

2. Problem solution approach
In the early 90’s several generations of Multivox TTS were developed, and as knowledge about text-to-speech systems with high naturalness increases, more generations are due to come. Therefore PDS must be flexible to changes in the TTS system, including changes in the format and number of data types used, the native TTS API, and even the synthesis technique used, and it must be able to handle multiple languages. This raises the need for a general development system architecture that gives support for common tasks and provides conveniences for extending the system with new modules.

The basics of PDS were formed 4 years ago [5]; since then the concept has matured and the system has been enriched with several parts.

The basic idea was to design such a modular system architecture that the interfaces of the components and their interactions are well specified. This architecture enables us to define mechanisms for adding new instances of the specified components to the system, this way new system behavior can be realized.

The TTS system is hidden in a server component behind a simple interface with only two basic operations (request, poke), and data is represented in an SGML based language called MVML (MultiVox Markup Language) [5]. MVML is different from STML [6] in the sense that STML contains tags for text description and speaker directives, while MVML also contains lower level commands for the synthesizer. Data can be MVML command sequences or data understandable to the TTS components. MVML commands are interpreted within a separate server component and translated into native TTS API calls. In order to make a TTS system usable with the development tool, the TTS system has to have the API functions necessary to implement the MVML commands. The server communicates with the client using DDE (Dynamic Data Exchange) under Windows, but the module implementing this can also be exchanged for another type of inter process communication.

The development system is a fully object oriented application with an intuitive graphic MDI (Multiple Document Interface) user interface. It behaves as a client to the synthesizer server. Both the TTS engine and the development system were implemented using platform independent code: the TTS engine uses standard C language,
the development system was created using wxWindows [7], a free C++ framework for cross-platform programming.

3. System architecture

The system is designed as multiple layers of modules. Each module consists of several interrelated classes. Each layer is based on the previous layer and modules in the layer are independent of each other, except for the Application layer, wherein they are interrelated. The hierarchy of the layers is shown in Figure 1.

![Figure 1: Hierarchy of layers in PDS](image)

1) Common layer 2) Service layer 3) Execution layer

The wxWindows classes form the very basis of the system, which determines some architectural decisions in areas related to the GUI; it also contributes some useful data structures, e.g., strings and hashed lists. The Common layer contains objects shared by the whole system, e.g., the resource classes, which store named values of various types in named collections, including graphic resources, strings, constants and user options. These resources can be saved to and loaded from external files.

The Service layer contains basic storage and visualization objects. These are used in the execution layer, whose objects define high-level components of the system, which have quite complex behavior.

The Application layer is made up of a few single-instance objects, which incorporate instances of the classes of the previous layer and organize them to form a solid application. The Voice Manager object keeps track of and manages the synthesizer voices used in the system. The Document Manager object is responsible for handling documents in the system. Documents are responsible for storing, displaying and manipulating data. Documents also have their own commands and menu bar. The Communications object can be referred to when operations concerning the server application are required. In the following key components of this architecture are discussed.

3.1. Architecture for working with voices

Operation with voices is one of the most important functions during speech research for TTS development. The notion ‘voice’ in our usage means the aggregation of a TTS synthesizer, a connection, a synthesis mode, a speech unit database or parameter database (henceforth database) and possibly dictionaries, sets of rules, etc. Documents for displaying or modifying properties of these are also associated to a voice.

Voices physically belong to the TTS servers, but they are mapped into the development system transparently, i.e. the way data is acquired is hidden. After setting up the connection, properties of the voice are acquired from the TTS server. Such properties are the list of phonemes used for the voice, the list of database elements in the database and physical properties of the database files. From this time on, we refer to the voice database as if it was actually part of the system. If the user wants to work with a database element, the system loads it from (stores it to) the database object, which gets it from (sends it to) the TTS server. Working with the database through the client/server architecture has the advantage that the development system does not need to know the database file formats.

The development system handles speech unit databases storing diphones, triphones or longer segments uniformly. A database element is identified to the server as the one belonging to an arbitrary length sequence of phonemes from the list of phonemes for the voice; other attributes of the element might also be specified to narrow the choice to one element if more elements exist for the same phoneme sequence, with e.g. ones different pitch. Phonemes are characterized by an ordinal number, an ASCII denotation and phonetic properties of the speech sound. The user can use either of the first two attributes to identify a phoneme, i.e. he or she can use a phonetic transcription or a numeric code of the database element. The phonetic properties are used by the system for algorithms that have to manipulate different kinds of sounds differently.

We solved the problem of being language independent by requesting language specific information from the synthesizer server. This does not essentially mean extra complexity because this information should be requested even if we wanted to support one language. For example the list of phonemes used in the database is characteristic of the synthesis language. But it is also characteristic of the database, and might well be different for two databases created for the very same language if a different concept is used.

3.2. Architecture for working with data

The development system requests some data from the connected TTS synthesizers: these are starting-point-, end-result- and intermediate-data produced at different phases of the synthesis process, as illustrated in the DreSS speech synthesizer tutorial [8]. In the Profivox speech synthesizer these data are marked up text, analyzed text, phonemic and signal representations of speech. These data formats are similar to the ones used by DreSS, with the difference that linguistic analysis is not included in the analyzed text format, instead more detailed sound level prosodic control codes are stored.

A data class can be used not only for synthesized speech data, but for internal types also, e.g. the results of a conversion of the speech signal into a parameter domain like LPC. The data class must contain everything related to the data type: fields for storing the information, functions for loading, saving and manipulating it, the associated file extension, the MVML identifier for sending it over a connection. The data object is also responsible for displaying the data in an appropriate form in the corresponding area of the document. Common functions and properties of data types are declared in the base class for data classes and reached through that. The set-up described above ensures that the system can be modularly expanded with new data types.

The relationship between synthetic data produced at different points of the synthesis can be preserved by the
There are several approaches regarding how this is to be stored.

Within a TTS engine, these data may be represented in a unified structure, as e.g. in the linguistic analysis strings of the Bell Labs TTS system [9]. This approach cannot work however within the development system, as different synthesizers may give data from more or less phases, and of several types. Therefore a general connection structure was used that can combine any kinds of data using numeric positions. All related data are stored within one document together with a collection of connections. The collection object can calculate the correspondence between segments of any two data representations. This information is used in the system for their parallel display and manipulation. Figure 2 shows the way text and the prosody matrix (sounds with physical parameter codes) are displayed parallel with the waveform for the Hungarian word \textit{kutya} ‘dog’.

### 3.3. Extending the system

The described modular architecture is especially useful when the need to extend the working system arises. The system can be extended into several directions. Figure 3 shows objects in the application layer with aggregated objects that are relevant from this point of view. Class names in bold indicate the points where an object with new behavior can be inserted.

The insertion of new components takes place in the service and in some cases in the execution layer. Since the base classes determine the relationship between the objects, in most cases it is sufficient to add the new objects and make an entry for them in some list in the application layer to make the extended system work, depending on the kind of change done.

The system is highly parametric since in all situations configuration files and lists are used instead of fixed data structures. This has the effect that in many cases one does not need to write new code when making minor changes to the system. Similarly, when information is requested or sent to the server application that can be described as a list of items in text format. This makes the system immune to a difference in the number of properties owned by different speech synthesizers or different versions of the same synthesizer, as the system is often insensitive to the presence or lack of a field.

If a new interprocess communication channel is applied for connecting to the TTS server, a new connection class is to be defined, which implements the basic communication operations (request, send) on the new protocol. The communication object refers to these operations of the connection object, which hides the actual implementation of the channel.

The most likely change to be made is adding support for working with new data types. This is done on two levels. When new speech related data is to be displayed and manipulated in a document, the first step is to define the necessary building blocks in the service level. This takes deriving a data class from the base class of data objects and defining necessary screen items for displaying it and processing user input. In the second step, the data must be integrated into the document, or a new document and the view for displaying it is to be derived. In both cases, defining the behavior of the data object is the majority of the work. This makes the extension of the system realizable without much extra effort.

A TTS system with a new type of synthesis (parametric or concatenative) can be added to the system by defining a new synthesizer object. The objects stores how the synthesizer is to be connected to, what synthesis method and database it uses and what kind of data it supplies.

When adding a new synthesis method, a new database type may also need to be defined if the elements to be stored are of a different type. This does not need to be done when only certain properties of the new database are different, like different number of items are stored or a triphone database is used instead of a diphone one.

### 4. Functions of the development system

The researcher can use the system to evaluate the quality of the implemented prosodic models and intonation algorithms.
in several ways. Intensity, frequency and time structure curves can be displayed for the synthesized waveform signals. The different prosodic components can be turned on or off individually, thus their contribution to the final result can be observed.

Artificial prosody can be applied to natural speech utterances, modifying only one intonation component at a time (pitch, rhythm or volume), this way the quality of each of these compounds can be estimated separately.

Various intermediate data can be acquired from the synthesizer and displayed in intuitive ways in the development system. The relationship between the different representations can be displayed by aligning the related data (e.g. text, phoneme codes, waveform data). Related data can be edited together as if they formed one unit: they can be deleted, cut, copied, or be pasted to a new location. Data at any phase of the synthesis process can be viewed and modified, e.g. the prosody matrix, and the synthesis process can be continued from that point with the modified intermediate data. This facility supports the development of new rules (prosody, timing, grapheme-sound conversion, etc.).

The TTS system developer can use the acquired intermediate and final speech forms and the connections between them to see if the implementation is in accordance with the specifications. Acquiring the values of internal states of the synthesizer is also possible.

There are multiple functions assisting the creation of new speech databases. The system can be used in a semi-automatic database creation process, in which the time structure of an existing database and appropriately named voice recording files are used to create a new speech database. The new database can be normalized for intensity. The typical intensity value of sounds can be calculated from a database, then the resulting values, possibly after a modification to fit theoretical values, can be used on the same or other databases to normalize them. Individual database elements (e.g. diphones, triphones) can be requested and stored to the database after modification. The element can be edited with the cut, copy and paste operations, and it can be scaled linearly or along predefined modification functions (raised cosine). The modifications can be tested and either be stored or be discarded, as desirable. The system can also be used as a labeling tool, i.e. for adding pitch marks with voiced/unvoiced information, sound boundaries and word boundaries to the waveforms. Statistics can be made of texts considering how many times each database element was used to synthesize the given text. These statistics can then be used to determine which database elements need more attention during development (e.g. one can see if an element did not occur in any of the texts used for auditive testing and still has to be checked).

5. Adaptation to new languages

The system, due to its flexibility and interactive working style supports tasks in relation with TTS development. The basic language independent structure makes it easy to develop new speech unit databases, rules for prosody and checking the operation of TTS modules for the new language. The messages of the system can be also easily adapted to another language, i.e. the proper text file has to be translated.

6. Conclusions and future directions

The described development system architecture gives a flexible framework for research, experimentation and development with a variety of TTS systems. The PDS system can be configured by parameter files and can be extended easily due to its modular architecture. The current implementation effectively helps the researcher, the teaching process as well as giving functions to facilitate the creation of speech databases and TTS application development and testing.

In the future, we want to prepare the application to work with other parametric synthesizers, too. We intend to add more data types and give more support for working with speech signals in the frequency and other parametric domains. Editing functions for dictionaries and rule-based parts of the synthesizer are also considered.

The authors are open to cooperation and adaptation to other languages.

7. Acknowledgements

This research was partly funded by the Hungarian National Committee for Technological Development, the National Telecommunications Fund and supported by Pannon GSM through a 10-month scholarship to Géza Németh and by Timber Hill LLC through a 6-month scholarship to Géza Kiss.

8. References