UPROOTING MARYTTS: AGILE PROCESSING AND VOICEBUILDING

Sébastien Le Maguer[1], *Ingmar Steiner*[1,2]
[1]*Saarland University,* [2]*DFKI GmbH*
{slemaguer, steiner}@coli.uni-saarland.de

Abstract: MaryTTS is a modular speech synthesis system whose development started around 2003. The system is open-source and has grown significantly thanks to the contribution of the community. However, the drawback is an increase in the complexity of the system. This complexity has now reached a stage where the system is complicated to analyze and maintain.
The current paper presents the new architecture of the MaryTTS system. This architecture aims to simplify the maintenance but also to provide more flexibility in the use of the system. To achieve this goal we have completely redesigned the core of the system using the structure ROOTS. We also have changed the module sequence logic to make the system more consistent with the designer. Finally, the voicebuilding has been redesigned to follow a continuous delivery methodology. All of these changes lead to more accurate development of the system and therefore more consistent results in its use.

1 Introduction

MaryTTS is a modular text to speech (TTS) synthesis system whose development started around 2003 [1]. The system is open-source and has grown significantly thanks to the contribution of the community.[1] However, the drawback is an increase in the complexity of the system. This complexity has now reached a stage where the system is complicated to analyze and maintain.

In this paper, we present the new architecture of the MaryTTS system. The main objective of this new architecture is to simplify the extension and the maintaining of the system. To achieve this goal, we need to solve three main problems. Firstly, we need to have a flexible and uniform – but accurate – *representation* of the information inside the system. Secondly, we need to extend the modularity of the system by making *modules* more independent and flexible. Finally, we need to provide a refined *process* to add a new synthesis voice. To achieve these goals, we mainly rely on the introduction of ROOTS into the MaryTTS system, as ROOTS provides an easy way to extend and adapt the data representation.

This paper is structured as follows. First the current architecture and problems related to it are described in Section 2. Then ROOTS and its integration into MaryTTS are presented in Section 3. Section 4 focuses on the new module sequencing paradigm and Section 5 on the new voicebuilding process.

2 Current architecture

2.1 Description of the current architecture

As described in Figure 1, the current architecture is relies mainly on four distinct concepts: the data, the datatypes, the modules, and the targets.

[1] `https://github.com/marytts/marytts`

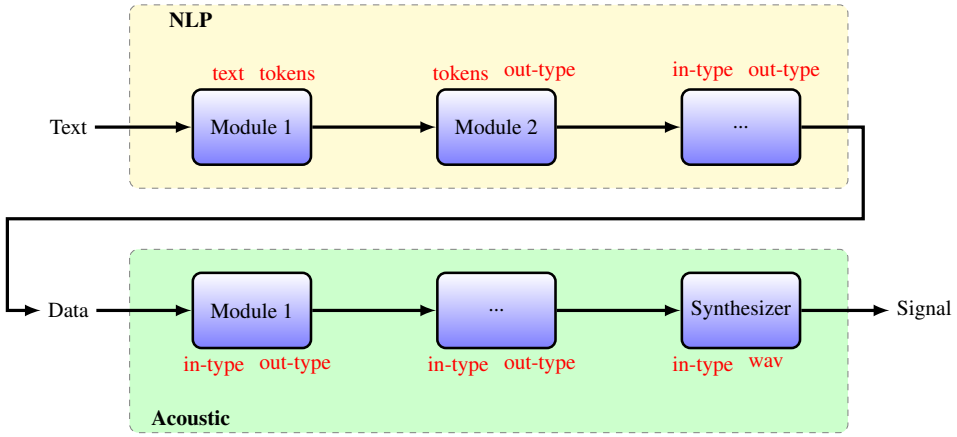

Figure 1 – Sequence of modules architecture of the MaryTTS system

The module is the key concept of the system. A module performs one operation in the synthesis pipeline. It can be at any level, either linguistic (e.g., part of speech (POS) prediction), prosodic (e.g., F0 prediction), or purely acoustic (e.g., unit selection). The goal of a module is to enrich the representation of the utterance by adding some information. Therefore a module takes a certain state of information as the input and produces another type of information as the output.

The type of information is referred as "datatype". A datatype is just a label which describes the current state of the description of an utterance. For example, "PHONEME" indicates that the utterance has been phonemized. This also means that all operations, and therefore all the modules, needed to reach this state, have been processed. For example, for the English language, the current sequence of modules leads to the "PHONEME" datatype.

To store the data, a custom XML format ("MaryXML") is used; an utterance is represented by an XML document. Therefore, the goal of the module is to enrich the document by adding more nodes or adding more attributes. The datatype corresponds thereby to the labeling of the current XML information; e.g, "PHONEME" implies the presence of "t" nodes with "ph" attributes, which correspond to the phonemization description of the text tokens.

Finally, for the machine learning and the acoustic prediction modules, the targets are introduced. For now, the target is a meta concept containing an XML element (for example, the segment for the acoustic prediction modules) and a feature vector. The key idea behind the target is to link the information used to generate the feature vector and the vector itself. Features are computed by processing the XML document from a specific segment (which is the baseline for the target element). New features can be introduced by overriding the feature processor class.

2.2 Drawbacks

The actual architecture leads to several drawbacks. First, the XML format is a documentation representation standard but it is tree based. Therefore, a significant processing overhead is introduced when the system needs to link two kinds of information widely spread across different levels. Furthermore, the use of the same format makes it difficult to extend the information and therefore limits the overall extensibility of the system. Indeed, the relation between elements is not always an inclusion of multiple elements or a specification of one attribute. For example, the word "it's" is associated to two POS tags. For now a meta unit is created but this is more a hack than a real extensible solution.

Secondly, the notion of datatype forces the user to know which type of information is produced by each module. This also restricts the flexibility of the system as the developer of a new module may have to define a new datatype. More generally, the user tends to define a sequence of modules and might replace the use of one specific module by another. Therefore, the way of defining a sequence of modules in MaryTTS is not natural.

Finally, the current voicebuilding framework relies on a GUI application whose drawbacks include accessibility issues, as well as a lack of portability, explicit processing order, parallelization, caching of completed steps, and various issues related to external program invocations.

For all these reasons and as a lot of developers participated to the evolution of MaryTTS, the system has become complicated and difficult to maintain. Our goal is to simplify the system and refactor the platform core to make it easier to maintain.

3 Introducing ROOTS

The first major change concerns the replacement of the data representation inside the system itself to use a lightweight implementation of ROOTS [2, 3]. To describe this replacement, we are first going to describe ROOTS and then explain how it is integrated in MaryTTS.

3.1 Description of ROOTS

The core of ROOTS is based on three essential concepts: the *item* which represents the actual information; the *sequence* that contains items and the *relation* that enables linking of items in two sequences.

An item is an instance of information. A word, a phoneme, a syllable are all examples of items. In order to be the more flexible, the item definition relies heavily on the object programming paradigm. Therefore, based on subclassing and polymorphism, it is possible to extend easily available information in the system. For example, let's consider a phone an extended phoneme; it is extended by adding the start/end position in a corresponding signal. Therefore, to define a phone, we simply subclass the phoneme and add the information {`start`, `end`, `signal_filename`}.

A sequence contains items of a specific levels in sequential order. The generic aspect of the representation enables us to keep a coherent semantic for sequences by guaranteeing the homogeneity of items they contain. Consequently, we can talk about a sequence of words or a sequence of phonemes. However, the use of subclassing allows us to diversify the content of a sequence. Indeed, a sequence of phonemes can be composed also by phones. Consequently, each level is represented by one or more sequences allowing for several representations.

If the sequence allows to link items from the same levels, the relation enables the linking of items at different levels. A relation is represented in the form of a sparse matrix as shown in Figure 2a. A relation is oriented and therefore has a source sequence and a target sequence. By convention, the source sequence is represented by rows and the target by columns.

There are three main advantages of this representation. First, it is a natural way of reading this information. Secondly, obtaining the reciprocal relation is achieved easily by transposing the matrix representing the relation. Finally, let's assume we have two relations R_a^b and R_b^c but not R_a^c. We can obtain R_a^c easily by composing R_a^b and R_b^c using the matrix product $R_a^b R_b^c$. An example of this is illustrated in Figure 3.

There is one problematic case which is the composition which leads to an ambiguous relation like illustrated in Figure 4. The disambiguation is not solved by the system but it is assume to be the responsibility of the user, or of the module developer in the MaryTTS case.

ROOTS is an elegant and flexible way of structuring the information in the MaryTTS context

$$\begin{array}{c} \\ syl_1 \\ syl_2 \\ syl_3 \end{array} \begin{array}{c} \begin{array}{cccccccc} \text{w} & \text{V} & \text{n} & \text{s} & @ & \text{p} & \text{A} & \text{n} \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

(a) Example of relation $R_{syl}^{phonemes}$

$$\begin{array}{c} \\ \text{w} \\ \text{V} \\ \text{n} \\ \text{s} \\ @ \\ \text{p} \\ \text{A} \\ \text{n} \end{array} \begin{array}{c} \begin{array}{ccc} syl_1 & syl_2 & syl_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

(b) Example of the reciprocal relation of $R_{syl}^{phonemes}$

Figure 2 – Relation example for the text “Once upon”

$$\begin{array}{c} \\ \text{w} \\ \text{V} \\ \text{n} \\ \text{s} \\ @ \\ \text{p} \\ \text{A} \\ \text{n} \end{array} \begin{array}{c} \begin{array}{ccc} syl_1 & syl_2 & syl_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \end{array} * \begin{array}{c} \\ syl_1 \\ syl_2 \\ syl_3 \end{array} \begin{array}{c} \begin{array}{cc} \text{Once} & \text{upon} \end{array} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \end{array} = \begin{array}{c} \\ \text{w} \\ \text{V} \\ \text{n} \\ \text{s} \\ @ \\ \text{p} \\ \text{A} \\ \text{n} \end{array} \begin{array}{c} \begin{array}{cc} \text{Once} & \text{upon} \end{array} \\ \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \end{array}$$

Figure 3 – Composition of a relation linking the phonemes to the syllables and a relation linking the syllables to the words. The result relation is linking the phonemes to the words.

$$\begin{array}{c} \\ \text{I} \\ \text{t} \\ \text{s} \end{array} \begin{array}{c} \begin{array}{c} \text{it's} \end{array} \\ \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \end{array} * \begin{array}{c} \\ \text{it's} \end{array} \begin{array}{c} \begin{array}{cc} \text{it} & \text{is} \end{array} \\ \begin{pmatrix} 1 & 1 \end{pmatrix} \end{array} * \begin{array}{c} \\ \text{it} \\ \text{is} \end{array} \begin{array}{c} \begin{array}{cc} \text{Noun} & \text{Verb} \end{array} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array} = \begin{array}{c} \\ \text{I} \\ \text{t} \\ \text{s} \end{array} \begin{array}{c} \begin{array}{cc} \text{Noun} & \text{Verb} \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \end{array}$$

Figure 4 – Problematic composition of relations: an ambiguity is introduced as the composition of the relation $R_{\text{I t s}}^{\text{it's}}$ with the relation $R_{it's}^{\text{it is}}$ leads to put everything in relation. This is correct for this intermediate relation however, composing it to get the part of speech tag leads to a decision which needs to be taken.

as the system is assumed to be modular. Furthermore, at the opposite of Heterogeneous Graph Relation [4] (the structure implemented in Festival [5]), the redundancy of the information is maintained to keep this information available at all times. This suits perfectly the modularity design of MaryTTS, as it allows us to introduce a new module easily. Finally, the representation is independent of the serialization format. Therefore, it is possible to export the data in any kind of format, such as MaryXML.

3.2 Integration of ROOTS into MaryTTS

To integrate ROOTS into MaryTTS, a lightweight version of it has been implemented in java. This leads to a distinction of the inner representation of the data in MaryTTS and the representation used to feed the system.

As mentioned before, internally all the references to the XML format have been stripped out. This means that from one module to another, an utterance is passed in the form of a “plain

old java object". This also means that the XML routines have been removed and replaced by dedicated ROOTS routines.

Furthermore, the feature representation is also integrated into the ROOTS datastructure. This means that the concepts of target, which currently encapsulates an XML element (representing a segment) and feature vector are disappearing. Indeed, a target can be represented by a relation between the sequence of segments and the sequence of feature vectors.

In order to be able to feed the system, it is necessary to implement serializers. Currently, an XML serializer has been implemented to maintain backward compatibility with the previous version of MaryTTS. However, the XML document is an under-specification of a ROOTS utterance. This means that the compatibility is maintained with the standard modules of the systems but might not be supported with new modules which require more complex informations.

Even if it seems to be a drawback at first, processing this way is actually more flexible. Indeed once a proper serializer is developed, it is possible to plug in virtually any kind of format into the systems (e.g., Praat TextGrids, JSON, ...).

4 Module sequencing

Based on the new data representation paradigm, and for the previously mentioned reasons, the concept of datatype is disappearing from MaryTTS.

Therefore, the goal of a module is to take an utterance and to enrich it. Of course, it is possible that the information needed by a module is not available. However, we consider it the responsibility of the module to check the availability of this information.

Consequently, the processing pipeline is now defined by a sequence of modules that needs to be explicitly provided. Currently, we are considering three main checkpoints in the system: the language package, the acoustic generation package (unit-selection, HSMM, ...), and the voice. Each language package defines a default process from the text to the phonemization process. Each acoustic generation package defines a default process from the phonemization to the acoustic signal. A voice is defined according to a language and an acoustic generation method. For example, we can consider an English voice for unit selection. If nothing more is defined, the pipeline will be composed by the default modules indicated in the language and the acoustic generation packages. However, it is possible to define a new module sequence specifically for this voice in its configuration.

The introduction of this change makes the MaryTTS system even more flexible and "plug and play", which is ideal for a research tool. For now, the sequence is defined offline, but the core is present in the system. Once online sequencing is supported, MaryTTS will be ideal to compare different ways of running one stage of the process, as the rest can be kept completely constant.

5 A new voicebuilding process

We now rely on the *Gradle* build automation platform.[2] This leads to the integration of a new strategy in the voicebuilding process. We split the process into three main parts: descriptive feature extraction, acoustic parameter extraction, model generation.

The old "VoiceImportTools" with their custom GUI and ad-hoc design patterns had the right intuition to the process, but failed to provide a robust, efficient implementation of task-based workflows. In no particular order, the process was missing,

portability all generated file paths were absolute, so voice project directories could not be

[2] `https://gradle.org/`

moved without manual reconfiguration,

a task dependency graph components needed to be run in an implicit order by consulting the documentation,

caching re-running a components processing method would always repeat previously successful work,

logging messages were simply printed to standard output,

parallelism only one component could run at a time,

consistent subprocess management external tools could fail to run properly,

principled testing only some components contained inline assertions,

stable packaging resources were be copied into a new project structure, where Maven would be invoked in a subprocess.

These shortcomings were clear, and some vague design notes formulated by Marc Schröder[3] are almost prophetic in the way the requirements are – nearly perfectly – met by Gradle's features and capabilities.

We have reimplemented the voicebuilding process as a collection of Gradle plugins which can run the workflow end-to-end with maximal efficiency.[4] By tapping into Gradle's feature set, we can benefit from,

- cross-platform builds with "lazy" path and property evaluation,
- a task dependency model based on a directed acyclic graph (cf. Figure 5)
- task output caching,
- a logging engine,
- parallel task execution,
- robust logic for external command execution and Java subprocess management,
- flexible testing and report aggregation,
- dependency resolution and artifact publishing engines,
- a powerful domain-specific language based on Groovy,
- extensibility and customization via plugins.

This also allows us to efficiently run validation experiments and deploy voices in a continuous delivery paradigm.

The benefits of Gradle have also been introduced for the primary build platform of MaryTTS, as well as other custom build tasks, such as letter-to-sound resource and source generation for new language support [6].

[3] In June 2012, shortly after MaryTTS development had moved to GitHub, Marc Schröder wrote into the newly-created wiki at `https://github.com/marytts/marytts/wiki/Ideas-for-future-work`:

> **Voicebuilding**
>
> aim: introduce robustness, transparency (so you know what's going on) and the ability to parallelize into the process.
>
> idea: insert the concept of a "data item" which knows its current status.
>
> A data item object represents the output or input of a processing step. For example, a data item can represent a wav file, or a pitchmark file, i.e. a certain processing result for an individual basename; but it can also represent an agglomerative output, such as a waveform timeline or a duration prediction tree.
>
> Each component has a list of required data items and a list of produced data items.
>
> This means for the data item "unit selection voice" we can look backwards how we can build this.
>
> For simplicity, our data item sequence is predefined, not "emerging" as in MARY TTS. That is, the sequence of steps is clear; the components implementing the steps can be exchanged or selected (e.g., several possible pitchmarkers).
>
> At startup, the database layout checks which data items are up to date.

[4] `https://github.com/marytts/gradle-marytts-voicebuilding-plugin`

6 Conclusion

In conclusion, we have presented the new architecture of the MaryTTS system. This architecture revitalizes the idea that MaryTTS is a TTS system designed with modularity in mind. Therefore, the new architecture aims to extend this idea and make the system more flexible.

To this end, we have integrated a lightweight implementation of ROOTS, a flexible representation framework for speech and linguistic data. We have also redesigned the module sequencing in order to make it more natural to use. Finally, we have developed a new voicebuilding process which adopts the paradigm of continuous delivery.

To extend the work presented here, some improvements of the system are currently underway. First, we are currently integrating the latest version of the statistical parametric synthesis methodologies into the system. We may then integrate a hybrid methodology. We also plan to add an online module integration support. This will lead to a full plug and play TTS system. Finally, we are considering to replace the default phone representation, based on X-SAMPA, by an internal IPA representation. Similarly to the data serialization, a phonetic alphabet converter will be provided, which can also be extended. This change will also facilitate the integration of a new language in the system.

References

[1] SCHRÖDER, M. and J. TROUVAIN: *The German text-to-speech synthesis system MARY: A tool for research, development and teaching*. *International Journal of Speech Technology*, 6(4), pp. 365–377, 2003. doi:10.1023/A:1025708916924.

[2] CHEVELU, J., G. LECORVÉ, and D. LOLIVE: *ROOTS: a toolkit for easy, fast and consistent processing of large sequential annotated data collections*. In *International Conference on Language Resources and Evaluation (LREC)*. 2014. URL `http://www.lrec-conf.org/proceedings/lrec2014/summaries/338.html`.

[3] BARBOT, N., V. BARREAUD, O. BOEFFARD, L. CHARONNAT, A. DELHAY, S. LE MAGUER, and D. LOLIVE: *Towards a versatile multi-layered description of speech corpora using algebraic relations*. In *Interspeech*, pp. 1501–1504. 2011.

[4] TAYLOR, P., A. W. BLACK, and R. CALEY: *Heterogeneous relation graphs as a formalism for representing linguistic information*. *Speech Communication*, 33(1-2), pp. 153–174, 2001. doi:10.1016/S0167-6393(00)00074-1.

[5] BLACK, A., P. TAYLOR, R. CALEY, R. CLARK, K. RICHMOND, S. KING, V. STROM, and H. ZEN: *The Festival speech synthesis system, version 1.4.2*. 2001. URL `http://www.cstr.ed.ac.uk/projects/festival/`.

[6] STEINER, I., S. LE MAGUER, J. MANZONI, P. GILLES, and J. TROUVAIN: *Developing new language tools for MaryTTS: the case of Luxembourgish*. In *28th Conference on Electronic Speech Signal Processing (ESSV)*, pp. 186–192. Saaarbrücken, Germany, 2017.

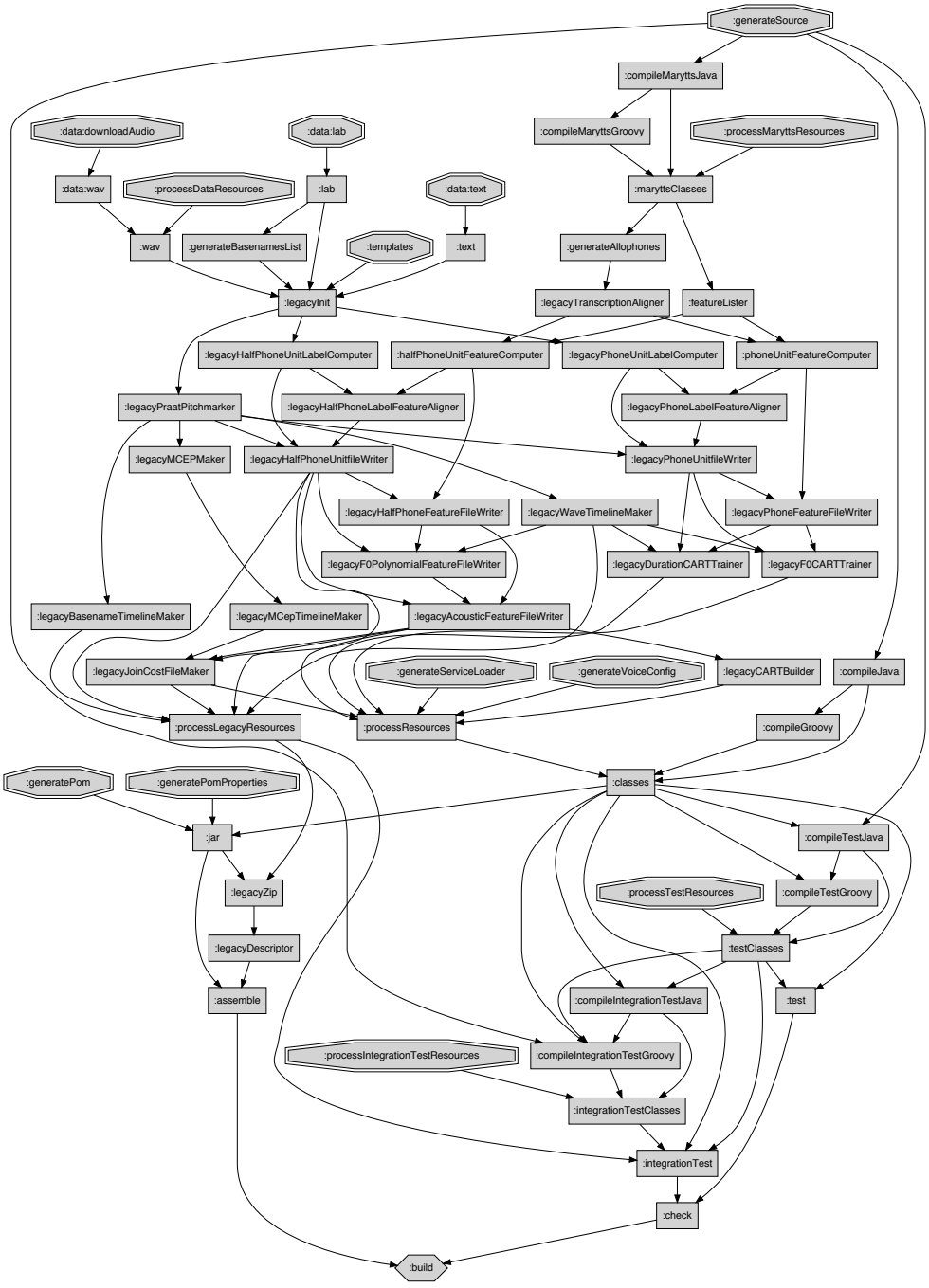

Figure 5 – Task execution graph using the Gradle-based voicebuilding plugin to create a new MaryTTS synthesis voice. Tasks with names starting with "legacy-" wrap the corresponding old components.