Creating New Language and Voice Components for the Updated MaryTTS Text-to-Speech Synthesis Platform

Ingmar Steiner, Sébastien Le Maguer
Saarland University & DFKI GmbH
{steiner, slemaguer}@coli.uni-saarland.de

Abstract

We present a new workflow to create components for the MaryTTS text-to-speech synthesis platform, which is popular with researchers and developers, extending it to support new languages and custom synthetic voices. This workflow replaces the previous toolkit with an efficient, flexible process that leverages modern build automation and cloud-hosted infrastructure. Moreover, it is compatible with the updated MaryTTS architecture, enabling new features and state-of-the-art paradigms such as synthesis based on deep neural networks (DNNs). Like MaryTTS itself, the new tools are free, open source software (FOSS), and promote the use of open data.

Keywords: text-to-speech synthesis (TTS), front-end, multilingual

1. Introduction

Over the last 15 years, MaryTTS (Schröder and Trouvain, 2001) has become one of the reference systems for open source text-to-speech synthesis (TTS). Today, it is actively used by researchers working in speech science, human-computer interaction (HCI), and related fields, as well as by professional and enthusiast software developers in free, open source software (FOSS) or enterprise settings. Its popularity is due in part to the number of languages and voices which are freely available as open resources, as well as the possibility of extending it to support new languages and building custom synthetic voices, or even integrating MaryTTS as a component into more complex applications, such as TTS web services, accessibility software, or spoken dialog systems (SDSs). Because of its implementation in the Java programming language, MaryTTS can be used on any device or computer with a Java Runtime Environment (JRE), and its modular design allows developers and users alike to inspect and customize the entire processing pipeline from input text to speech output.

However, the number of people who have participated in, and contributed to, MaryTTS development over the years has led to a complex and overburdened system. Consequently, a reboot of the system became unavoidable; until now, we focused on restructuring the system core and explained the philosophy behind the new architecture (Le Maguer and Steiner, 2017a; Le Maguer and Steiner, 2017b).

Independently, the process of creating new synthetic voices and support for new languages in MaryTTS has also fundamentally evolved since it was presented by Pammi et al. (2010). Therefore, the current paper presents the new language and voice building workflow for MaryTTS.

The remainder of the paper is structured as follows. Section 2 provides a brief background on build automation in MaryTTS. In Section 3, we present the new workflow to add support for a new language. Then, in Section 4, we focus on the new voice building pipeline. Finally, in Section 5, we present the reorganized source code and project hosting, particularly from a user perspective.

2. Background

Development on MaryTTS has adopted several significant paradigms which had become best practice in Java-based software engineering in the years since the project's inception. These include,

dependency management, where required software libraries are downloaded from cloud-based repositories,[1]

software testing, and

convention over configuration, where common standards are integrated into the software build lifecycle without the need for redundant specification.

In the latest version of MaryTTS, all of these aspects are managed through the *Gradle* build automation tool.[2]

The increase in flexibility and efficiency provided by Gradle is not limited to the development "under the hood". Rather, we leverage Gradle as a user-facing tool which replaces the custom applications previously required to add new languages to MaryTTS, or build new synthetic voices. This shift removes numerous limitations on performance and functionality, and solves common, recurring problems with installing third-party tools and writing boilerplate code for new MaryTTS components. At the same time, the text and speech data itself — required to build new components — can be managed as dependencies, and the components can be built, tested, and distributed more efficiently.

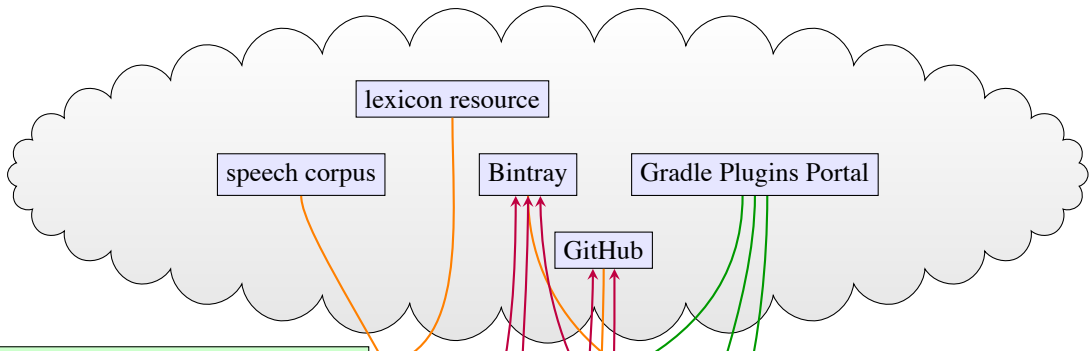
An overview of the entire workflow to create new language and synthetic voice components is shown in Figure 1. However, this workflow can be broken up into several independent steps, which are described in the following sections.

3. New Language Support

The purpose of a language component in MaryTTS is to allow the system to extract linguistic features from orthographic text using natural language processing (NLP). This includes, at the very least, the sequence of phonemes, i.e., the pronunciation, but typically also other features related to

[1] Examples of such dependencies in MaryTTS include third-party libraries for text tokenization (JTok), number expansion (ICU4J), and part-of-speech (POS) tagging (OpenNLP).

[2] https://gradle.org/



obtains lexicon resource
applies `lexicon-compiler-plugin`
trains lexicon FST and G2P rules
publishes `marytts-lexicon-xy`

(a) The lexicon project, which resolves or contains lexicon resources and is published to Bintray.

depends on `marytts-lexicon-xy`
provides NLP modules for language "xy"
publishes `marytts-lang-xy`

(b) The language project depends on the lexicon project, may contain other modules, and is published to Bintray.

obtains audio and text resources
obtains phonetic annotation *or*
applies `kaldi-mfa-plugin`
publishes
`speaker-somename-xy-data`

(c) The data project resolves audio and text from a speech corpus and either converts provided phonetic annotation or runs forced alignment using the language project. It can be published to GitHub as a release asset.

depends on `marytts-lang-xy` and
`speaker-somename-xy-data`
applies `voicebuilding-plugin`
1. extracts acoustic features from audio
2. extracts linguistic features from text
3. aligns features based on phonetic labels
4. builds models
5. packages data (for unit selection)

publishes `voice-somename-xy`

(d) The voice project resolves the processed data from the cloud (e.g., downloading from GitHub), runs all steps required to build a voice, and is published to Bintray. Large unit selection voice packages can be published to GitHub as release assets.

Figure 1: Overview of the complete workflow for a new language "xy" and synthetic voice components. Dashed blue arrows visualize the dependency of the voice project (Figure 1d) on the language and data projects (Figures 1b and 1c, respectively), the dependency of the data project on the language project, and the language project on the lexicon project (Figure 1a). All of these depend on the core MaryTTS runtime libraries (not shown), which are resolved from Bintray. Orange and purple arrows show the actual dependency resolution from, and publishing to, cloud-hosted services, respectively. Green arrows show plugins resolved from the Gradle Plugins Portal. Note that all or part of the cloud-hosted infrastructure (shown inside the cloud) could also be replaced by internal, non-public repositories.

phonology and used for the prediction of acoustic parameters, such as segment duration and fundamental frequency (F_0). Pronunciation prediction in MaryTTS is handled by a language-specific "Phonemiser" module, which looks up each text token in a lexicon and returns the sequence of phonemes. For any out-of-vocabulary (OOV) tokens, the module falls back to rules for grapheme-to-phoneme (G2P) prediction.

To add support for a new language to MaryTTS, the first step is to define the set of phonemes to be used, along with their standard phonological features, based on the International Phonetic Alphabet (IPA). The next step is to obtain (or create) a lexicon resource, ideally a text file, spreadsheet, etc., containing a list of words with their orthographic and corresponding phonetic transcription.

Finally, the lexicon is automatically compiled into a finite state transducer (FST)-based representation, relying in part on the WEKA toolkit (Hall et al., 2009). In the past, this was done using the custom *TranscriptionTool* GUI application (Pammi et al., 2010), which however suffers from various usability and performance issues. To improve this situation, we have developed a Gradle plugin[3] to convert the lexicon into the format required by MaryTTS. Furthermore, we are currently developing a more state-of-the-art G2P approach based on TensorFlow (Abadi et al., 2016), comparable to that of, e.g., van Esch et al. (2016).

It is possible to create further NLP modules for the new language component, handling text normalization to expand acronyms, numbers, and so on, into pronounceable repre-

[3]https://github.com/marytts/gradle-marytts-lexicon-compiler-plugin

sentations, POS tagging, etc. Alternatively, MaryTTS can just fall back to generic modules for such tasks. All of these modules are then combined to build the new language component, which will be used to process input text, and represents a dependency of the synthetic voice building, and ultimately, full TTS in the new language.

4. Voice Building

Building a new synthesis voice for MaryTTS consists of three distinct stages, (a) data preparation, (b) feature extraction, and (c) model building, which are described in the following subsections. All three stages are handled efficiently using Gradle plugins,[4] which wrap third-party tools and can run tasks in parallel where appropriate, speeding up the voice building process significantly compared to the old toolkit.

4.1. Data Preparation

When preparing the recording of speech data intended to create a new synthesis voice, it is common practice to create a prompt list which covers the phonetic (and possibly prosodic) inventory of the corresponding language, as well as the content of the voice's domain. These prompts are then read out by the voice talent over one or more recording sessions, preferably in a studio environment.

The previous voice creation toolkit (Pammi et al., 2010) promoted the use of a custom Java-based recording tool named *Redstart*, which is able to display a sequence of prompts on a computer screen and record the user reading them through the computer's microphone. While MaryTTS Redstart remains fully functional, it may not be usable in every recording scenario. For instance, in a professional recording studio, the voice talent is typically recorded using a digital audio workstation (DAW), and any visual presentation of prompts may only be possible using a separate computer. In other cases, the goal may be to record a more fluent performance (such as an audiobook), and a user experience that forces the voice talent to pause for each prompt would be too disruptive.

Regardless of which text prompts are selected, or how they are recorded, the outcome of this process is a set of text and audio files with corresponding contents. However, before these files can be used to build a synthetic voice for MaryTTS, they have to be phonetically annotated. This step requires determining the pronunciation of each text prompt, i.e., the sequence of phonetic units, and mapping them to the recorded audio's time domain; the process is related to automatic speech recognition (ASR), except that the expected content is known, and the sequence of phonetic units can be forced to align with the audio; this is known as *forced alignment*. In the past, the MaryTTS voice building tools relied on integrating third-party tools for this task, including *Sphinx-4* (Walker et al., 2004), *HTK* (Young et al., 2006), or the *FestVox* tool *EHMM* (Prahallad et al., 2006); however, MaryTTS users often report problems installing or running them, and errors are difficult to solve. More recently, *Kaldi* (Povey et al., 2011) has emerged as a leading ASR toolkit, and it has been integrated into the *Montreal Forced Aligner* tool (McAuliffe et al., 2017). This tool in turn has been integrated into the MaryTTS data preparation workflow in the form of a Gradle plugin.[5] The pronunciation can be predicted using MaryTTS and collected into a custom dictionary for Kaldi, then acoustic models are trained from the recorded data, and the phonetic unit boundaries are aligned and stored in the form of Praat TextGrids; this process is fully automated and can take a few minutes or hours, depending on the amount of recorded data.

Previously, the forced alignment process was described as part of the voice building process in MaryTTS (Pammi et al., 2010), but it can be more appropriately regarded as a prerequisite. While it is still possible to use both the forced alignment and voice building plugins in the same Gradle project, a more efficient workflow is to build a data *artifact*, which is then available as a dependency for the proper voice building process. Therefore, this stage can be skipped if a corpus of speech data is already available with appropriate orthographic and phonetic annotations.

4.2. Feature Extraction

At the core of the voice building process, the recorded speech data is converted to a *feature representation*. It is this feature representation which allows the use of machine learning techniques to train models to predict prosody and/or vocoder parameters from text during the actual TTS process in the runtime system.

The feature extraction stage of the voice building process yields a combination of frame-wise feature vectors from acoustic analysis of the audio, and time-aligned symbolic features based on linguistic analysis of the corresponding text; the alignment is based on the phonetic annotation obtained in the data preparation (cf. Section 4.1).

Acoustic features include F_0, tracked using Praat (Boersma, 2001), and mel-frequency cepstral coefficients (MFCCs), extracted using the Edinburgh Speech Tools (EST) (King et al., 2003). The linguistic features are obtained depending on the MaryTTS language component for the corresponding language. When creating a synthetic voice for a new language, this is where the new language component built previously (cf. Section 3) is used. The linguistic features extracted and assigned to the feature vectors include several related to phonology (e.g., distinctive features, position in the syllable, stress, accent), syntax (e.g., POS, distance to phrase and sentence boundaries), and — optionally — speaking style (Steiner et al., 2010; Charfuelan and Steiner, 2013), information density (Le Maguer et al., 2016), or other high-level context features.

4.3. Model Building

Depending on the underlying synthesis paradigm, it is possible to build a *unit selection* voice or a *statistical parametric synthesis* voice.

4.3.1. Unit Selection

Unit selection synthesis concatenates halfphone-sized snippets of natural speech selected from a voice database, given

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

[5] https://github.com/marytts/gradle-marytts-kaldi-mfa-plugin

target features computed for an input utterance. The output can sound very natural, but often suffers from audible glitches when synthesizing out-of-domain utterances, and prosody control is limited. Moreover, the voice database can be very large, as it contains the actual audio data.

Building a unit selection voice for MaryTTS involves storing the feature representation and related metadata for each unit, training statistical models for sparse prosody prediction, and packaging these along with the actual audio data. We have created a Gradle plugin which wraps some of the old toolkit's components to assemble unit selection voices which are backward-compatible with the current stable release of MaryTTS (v5.2). In addition, we are developing new build tools to support audio compression and enable prosody modeling and target feature prediction using hidden Markov models (HMMs) or deep neural networks (DNNs), paving the way for state-of-the-art "hybrid" TTS.

4.3.2. Statistical Parametric Synthesis

MaryTTS has supported statistical parametric synthesis for numerous years, using a Java port of the HMM based speech synthesis system (HTS) engine API[6] with a mel-generalized log spectrum approximation (MLSA) vocoder. Although such synthesis can sound rather buzzy and unnatural, these HMM-based voices offer higher flexibility and more consistent quality than unit-selection synthesis, as well as a much smaller memory footprint. However, some drawbacks are (a) that building HMM-based voices for MaryTTS has a high technical overhead, and (b) that the Java port has become quite outdated, while HTS development has seen significant progress. The former has been mitigated by providing a consistent, pre-configured *Docker* container, while to address the latter, we are developing completely new functionality. This includes the possibility to train models for third-party frameworks such as *Merlin* (Wu et al., 2016) and to allow other vocoders to be used, including *STRAIGHT* (Kawahara et al., 1999) or *WORLD* (Morise et al., 2016).

The parametric voice building process comprises three stages: the input and output feature packing, the model training, and the voice configuration generation. The voice configuration generation is similar to the unit selection voice building part (cf. Section 4.3.1). The output feature packing goal is just to adapt the acoustic features (e.g., mel-generalized cepstrum (MGC), F_0, band aperiodicity (BAP), etc.) to be compatible with the process used to train the models. Currently this means computing the delta and delta-delta coefficients and generating the binary observation vector for each utterance. The input feature packing consists of calling MaryTTS with a serializer dedicated to the training process.

The model training is a specific plugin implementing the process to train the models needed for the synthesis stage. We have developed a Gradle plugin dedicated to train HTS models (HMM-GMM or HMM-DNN).[7] This plugin can be adapted to the kind of parametric synthesis model or system we want to use.

[6] http://hts-engine.sourceforge.net/

[7] https://github.com/marytts/gradle-hts-voicebuilding-plugin

4.4. New Configuration Mechanism

Previously a configuration was attached to an artifact to configure the different modules. Moving forward, we consider three levels of configuration: the default configuration, the voice configuration, and the user configuration. The first of these is given in the module itself. The voice configuration corresponds to the parametrization of each module used during the voice building process and has priority over the default configuration. Finally, a user configuration can be specified at runtime, to override the other configurations.

5. Global Project Management

Refactoring the core system and of the voice building process has allowed us to separate the source code management (SCM) for each language and each voice project. Therefore, each language and voice can have its own SCM repository hosted on GitHub,[8] while the released artifacts are published to Bintray[9] and indexed in JCenter.[10] Any large data objects (specifically unit selection audio data) can be hosted on GitHub as release assets.

This makes the custom *Component Installer* GUI from previous MaryTTS versions obsolete, and allows us to replace it with a lightweight wrapper around the dependency management. A user can install and run MaryTTS voices and language components simply by executing Gradle tasks with the corresponding names; this is demonstrated by a new web installer for MaryTTS.[11]

Meanwhile, developers and researchers looking to integrate MaryTTS into their projects, only need to declare a dependency on the desired voice artifacts, and this will automatically resolve all transitive dependencies on the corresponding languages and other libraries.

6. Conclusion

In conclusion, we have presented a new language and voice building workflow designed for the updated MaryTTS system. We have detailed our reliance on the Gradle build automation tool, which provides a much more efficient and powerful framework via its extensible plugin system than the previous toolkit. We have also seen that the language components maintain the same concepts as in previous versions, but the methodologies used are updated. Finally, we have described the redesigned and extended voice building process, as well as our leverage of cloud-based infrastructure for hosting and distribution.

The next stage is to integrate the new MaryTTS core, state-of-the-art synthesis paradigms, and the new build system more deeply to provide the fully modular, modern TTS platform we are aiming for. Moreover, we are working to release the first preview of MaryTTS v6.0 in the coming months.

7. Acknowledgements

This work was funded by the German Research Foundation (DFG) under grants EXC 284 and SFB 1102.

[8] e.g., https://github.com/marytts/voice-dfki-spike

[9] https://bintray.com/marytts/marytts

[10] https://bintray.com/bintray/jcenter

[11] https://github.com/marytts/marytts-installer

8. Bibliographical References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–283, Savannah, GA, USA.

Boersma, P. (2001). Praat, a system for doing phonetics by computer. *Glot International*, 5(9/10):341–345.

Charfuelan, M. and Steiner, I. (2013). Expressive speech synthesis in MARY TTS using audiobook data and EmotionML. In *Interspeech*, pages 1564–1568, Lyon, France.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD Explorations*, 11(1):10–18.

Kawahara, H., Masuda-Katsuse, I., and de Cheveigné, A. (1999). Restructuring speech representations using a pitch-adaptive time-frequency smoothing and an instantaneous-frequency-based F0 extraction: Possible role of a repetitive structure in sounds. *Speech Communication*, 27(3-4):187–207.

King, S., Black, A. W., Taylor, P., Caley, R., and Clark, R., (2003). *Edinburgh Speech Tools Library*. Centre for Speech Technology, University of Edinburgh.

Le Maguer, S. and Steiner, I. (2017a). The 'uprooted' MaryTTS entry for the Blizzard Challenge 2017. In *Blizzard Challenge*, Stockholm, Sweden.

Le Maguer, S. and Steiner, I. (2017b). Uprooting MaryTTS: Agile processing and voicebuilding. In *28th Conference on Electronic Speech Signal Processing (ESSV)*, pages 152–159, Saarbrücken, Germany.

Le Maguer, S., Möbius, B., and Steiner, I. (2016). Toward the use of information density based descriptive features in HMM based speech synthesis. In *8th International Conference on Speech Prosody*, pages 1029–1033, Boston, MA, USA.

McAuliffe, M., Socolof, M., Mihuc, S., Wagner, M., and Sonderegger, M. (2017). Montreal Forced Aligner: Trainable text-speech alignment using Kaldi. In *Interspeech*, pages 498–502, Stockholm, Sweden.

Morise, M., Yokomori, F., and Ozawa, K. (2016). WORLD: A vocoder-based high-quality speech synthesis system for real-time applications. *IEICE Transactions on Information and Systems*, E99-D(7):1877–1884.

Pammi, S., Charfuelan, M., and Schröder, M. (2010). Multilingual voice creation toolkit for the MARY TTS platform. In *7th International Conference on Language Resources and Evaluation (LREC)*, pages 3750–3756, Valletta, Malta.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlíček, P., Qian, Y., Schwarz, P., Silovský, J., Stemmer, G., and Veselý, K. (2011). The Kaldi speech recognition toolkit. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, Waikoloa, HI, USA.

Prahallad, K., Black, A. W., and Mosur, R. (2006). Subphonetic modeling for capturing pronunciation variations for conversational speech synthesis. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume I, pages 853–856, Toulouse, France.

Schröder, M. and Trouvain, J. (2001). The German text-to-speech synthesis system MARY: A tool for research, development and teaching. In *4th Speech Synthesis Workshop (SSW)*, Perthshire, Scotland.

Steiner, I., Schröder, M., Charfuelan, M., and Klepp, A. (2010). Symbolic vs. acoustics-based style control for expressive unit selection. In *7th Speech Synthesis Workshop (SSW)*, pages 114–119, Kyoto, Japan.

van Esch, D., Chua, M., and Rao, K. (2016). Predicting pronunciations with syllabification and stress with recurrent neural networks. In *Interspeech*, pages 2841–2845, San Francisco, CA, USA.

Walker, W., Lamere, P., Kwok, P., Raj, B., Singh, R., Gouvea, E., Wolf, P., and Woelfel, J. (2004). Sphinx-4: A flexible open source framework for speech recognition. Technical Report TR-2004-139, Sun Microsystems.

Wu, Z., Watts, O., and King, S. (2016). Merlin: An open source neural network speech synthesis system. In *9th Speech Synthesis Workshop (SSW)*, pages 218–223, Sunnyvale, CA, USA.

Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X. A., Moore, G., Odell, J., Ollason, D., Povey, D., Valtchev, V., and Woodland, P., (2006). *The HTK Book (for HTK Version 3.4)*. Cambridge University Engineering Department.