# ARTIC for Assistive Technologies: Transformation to Resource-Limited Hardware

Daniel Tihelka, Petr Stanislav

***Abstract*—The present paper describes the transfer of ARTIC TTS to the real smartphone/communicator hardware, as the increasing penetration of such devices provides the solid base for the creation of special assistive technology applications, rendering the communication with hearing- or speaking-impaired people markedly more effective, simple and natural. As the target platform, the Google's Android OS has been chosen. In the paper, the unique system software stack of the Android OS is described, and the specifics of low-resource hardware are discussed from the point of view of two ARTIC TTS technologies single unit instance and multiple unit instance (also known as unit selection). The performance of the ARTIC running on the given low-resource hardware is presented and discussed.**

***Index Terms*—speech synthesis, assistive technologies, low-resource devices, Android OS.**

## I. Introduction

The ongoing increase in low-powered CPU speeds and memory sizes, followed by lowering their prices, results in the increased interest in smartphones and communicators. The first massive penetration was with the introduction of Apple's iPhone, later followed by T-Mobile G1, on which Android [1], the new Linux-based OS maintaining by Google, was used. The Android OS very soon gained popularity and more and more smartphones appeared on the market.

The fairly acceptable price of such devices allows hearing- or speaking-impaired or disabled people to purchase them and use them as a valuable helping tool that makes their everyday communication with the rest of the population far more effective, natural and pleasant. In addition to our experience with (audio-)visual speech synthetis [3], [4] created to help hearing-impaired people, we are planning to create a special assistive application (running on the majority of today's most common embedded platforms) for people with total laryngectomy surgery interposition. It will combine predictive keyboard based on LVCSR language modelling, adapting itself during its usage, with speech synthesizer immediately reading what is written. The aim is to allow more comfortable, intelligible, and to some extent also natural communication when compared to the use of electrolarynx device. However, to create such assistive tools, which are strictly language-dependent, we need a TTS system generating close-to-natural speech while leaving enough hardware resources for the application core and the rest of system, which also means not draining battery power almost only for running the TTS.

The fast-growing interest in smartphones presents, in general, a good opportunity for TTS systems, as they can be used not only for assistive technologies (although they are clearly the most meritorious), but also by text/SMS readers, navigation software and other tools allowing to detach the visual attention from the device in cases when it needs to be focused somewhere else (i.e. when driving a car).

In the paper, we have focused on the Android OS. Its choice was primarily due to the most limited support for C/C++ development (significantly wider range of C++ standard is supported by iOS or Windows CE/Mobile/Phone 7), which allows us to verify the platform independence of ARTIC TTS [2] as well as to identify obstacles to be solved before any assistive application can begin to be worked on. An additional - while unplanned - benefit is the ever-growing popularity of devices operating by the OS — according to [8], 36.4% of the new smartphones subscribed in the United States from January 2011 to Appril 2011 were powered exactly by the Android OS. The popularity is also increased by the ever-growing number of third party applications (over 150,000 application up to 10.2010) available for all the users of the OS through the on-line software store. In addition, the Android runs on wide range of devices from various manufactures

The transformation of ARTIC TTS to a representative communicator device operated by the Android is described in the present paper. We describe the unique system software stack of the Android OS with the emphasis of specifics and limitations of C/C++ development and their influence on the TTS build process. Further, the specifics of low-resource hardware are discussed from the point of view of two TTS technologies single unit instance and multiple unit instance (also known as unit selection). The performance of the ARTIC running on a given low-resource hardware is presented and discussed with the focus on various aspects like speed and battery exhaustion.

## II. Android Platform

The Android is the operating system designed in particular for mobile devices like smartphones and tablets, currently developed and maintained by Google. It is based on modified Linux kernel (the modifications are related mainly to suspend/resume of OS) and GNU software system utilities (dhcpcd, wpa-supplicant, bluez, dbus, SQLite, etc.), accomplishing low-level system tasks. On top of this, Android's specific Java virtual machine, named Dalvik, provides for all the applications both interface to system components, and to Android's display manager and the GUI itself. The choice of Java as the main (and mostly the only) interface to the

Manuscript received June 28, 2011; revised August 11, 2011. This work has been supported by the Grant Agency of the Czech Republic, project No. GACR 102/09/0989 and by the internal UWB grant, project No. SGS-2010-054.

D. Tihelka is with the Department of Cybernetics, University of West Bohemia, e-mail dtihelka@kky.zcu.cz

P. Stanislav is with the Department of Cybernetics, University of West Bohemia, e-mail pstanisl@kky.zcu.cz

ISBN: 978-988-18210-9-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

system does thus not allow the development of purely native applications and custom system components. On the other hand, its effective fixing as the main programming language for the OS allowed the rapid development of the enormous number of $3^{\mathrm{rd}}$ party applications, mostly by independent individuals.

The drawback of this solution is that the porting of already existing non-Java applications has become much more complicated. Although there is *native development kit* (NDK) available for the development in C/C++, its very limited support of C/C++ features and interaction with the system may cause hard-to-solve issues. Namely, the following constraints must be taken into account for $\mathrm{NDK} <= \mathrm{rev}.4$:

- GNU *libc* does not exist on Android, a subset called *bionic* exists instead
- only limited support of C++ code, RTII and Exceptions are explicitly not supported (although the lack of exceptions support is mentioned for Android v1.5 only)
- standard low-level C++ libraries like STL are not supported
- ports of common libraries used on GNU-Linux are rare, and, if exist, they must be linked statically (cannot be installed system-wide)
- documentation is sparse, incorrect, and sometimes confusing
- building is carried out by Android's native cross-compiler toolchain, widely used tools like *CMake* are not supported

It is due to the fact that the NDK has not been intended as a full alternative to Android's Java development model. According to the documentation, the native code should only be used very rarely, typically for libraries with "self-contained CPU-intensive operations that don't allocate much memory".

As the ARTIC TTS engine is written in C++, those limitations may complicate, limit or even prevent the possibility of transforming the TTS under that platform. This is why we have chosen it, because simply being able to get ARTIC working under Android we will be able to get it working under all not-as-much limited platforms (e.g. iOS or various smartphone-Windows).

### A. ARTIC for the Android Platform

Although ARTIC TTS is written in C++, from the very beginning of its development it has been emphasised that non-standard features and third-party libraries should be avoided wherever possible. This decision has proved beneficial to us, as there are no parts of the engine with features not-supported by the NDK. The only exception is the support of exceptions, but using an unofficial alternative NDK build [14] of the same version has solved this issue without any extra work.

The largest amount of work was spent on the transformation of the building process since ARTIC employs the CMake building system. Since there is a wide range of possible ARTIC build configurations, rather than converting the building process to the NDK's native building tools, we have decided to create a new CMake toolchain tied to Android's NDK. This allows us to transfer the already tuned flexibility of the TTS build to the Android without any need to rewrite it.

The TTS interface functions were wrapped by very thin layer using JNI library (provided as the part of NDK) to be accessible from Java, in which the rest of the testing application was created. Finally, all the C++ sources were smoothly compiled using CMake with the alternative NDK-r4 tools to `.so` library with ARMv7-A architecture. Let us emphasize that the `.so` library with ARTIC TTS and JNI wrapper, and simple Java class making the bridge to the TTS library, can now be used in any application created for the OS. Moreover, we plan to integrate the ARTIC to be one of the native system's TTS engines.

### B. The Hardware for testing

We have chosen two common communicators for the ARTIC performance measures. The first was more than a year old HTC Desire, the second was rather new Samsung Galaxy S. Their hardware specification is listed in Table I.

TABLE I
THE HARDWARE SPECIFICATION OF THE COMMUNICATORS USED.

| | HTC Desire | Samsung Galaxy S |
|---|---|---|
| CPU | Qualcomm Snapdragon, QSD 8250 | Cortex-A8 |
| frequency | 1 GHz | 1 GHz |
| cores | 1 | 1 |
| RAM | 576 MB | 512 MB |
| battery | Li-Ion 1,400 mAh | Li-Ion 1,500 mAh |
| data storage | 4GB micro SD card, external | 8 GB internal Flash memory |

The testing application, including the ARTIC TTS library, was installed into the default applications location within phone's internal memory, the speech unit database files were copied to the data storage from which they were "mapped to memory" (`mmap()` function under Linux). The Android OS version 2.2 (Froyo) was installed on the device.

## III. ARTIC EXPERIMENTS

There are two speech synthesis technologies employed within the ARTIC TTS – *single instance unit* (SUI) and *multiple instance unit* synthesis. The SUI version uses one off-line pre-selected candidate for each speech unit (triphone) [2]. Speech signal of the candidates must then be modified according to the prosodic contours estimated by prosody generator module [9]. This version is primarily intended for low/lower-resource hardware, as the speech unit database fits into several megabytes of space — customizable from 430MB, depending on the units clustering level and speech compression methods chosen. On the other hand, the modifications of speech makes it sound more robotic [10].

On the contrary, the MUI version (also known as *unit selection*) uses for each unit (diphone) all the candidates recorded in the speech corpus [2], [11], or optionally their subset, and the best candidate used to create synthetic speech is chosen according to the actual structure of synthesized phrase. Apart from the huge size of speech unit database — depending on candidates reduction level [12], [13] it scales from 600 down to approximately 100MB. The selection algorithm itself needs, naturally, much more computing power than the SUI version. Its significant advantage is, however, that the quality of synthetic speech is nearly natural-sounding.

ISBN: 978-988-18210-9-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

### A. Performance Measure

The speech segment database used for the experiments was recorded by a male semi-professional speaker in special echo-less recording studio. It consists of 12,242 utterances yielding approximately 18.5hours of speech (without pauses) [15], [16]. The corpus was segmented into phones automatically [5], [6], and triphone SUI and diphone MUI speech segment databases were created from that data. When in use, the database is "mapped to memory" from the communicator's data storage memory.

The measurement of the performance of Android's ARTIC build was split into initialization and synthesis parts. The initialization measured the time required to index units in the speech units database, which depends on its size. The synthesis part measured the time required for the synthesis of 250 clauses (giving 314 phrases), which resulted in approximately 14minutes of synthetic speech. Each experiment (both initialization and synthesis) was repeated 5times.

For both parts, the battery consumption was measured as well. This measuring, however, was only approximate, as Android does not expose interface for exact battery level check; instead, it sends events when the battery level changes. Let us note that to be able to measure the power consumption related to the TTS run, the synthetic speech was not actually played (energy consumed by 30 seconds of synthesis will surely be lower than consumption of 10 minutes more of playback, no matter lower load of hardware). Therefore, the playback duration was computed from speech length.

### B. ARTIC SUI Version

The speech segment database for SUI version contained 5662 unique triphones and occupied approximately 20.4MB on data storage. Using more efficient compression, its size can be reduced down to 5MB. Further size reduction, down to about 1.6MB [7], can be achieved by more aggressive triphones clustering during the HMM segmentation phase, although the pressure to clustering may of course affect the quality of synthetic speech. Such reduction, nevertheless, would have positive effect on the initialization part.

Considering the named size, Table II collects the results of performance measured on both devices. It can be seen from

TABLE II
THE PERFORMANCE OF ARTIC SUI INITIALIZATION AND SYNTHESIS. THE "REALTIME" IS THE DURATION OF SYNTHESIZED SPEECH. MEDIAN OF INITIALIZATION TIME IS PRESENTED INSTEAD OF MEAN, SINCE MEAN VALUE IS MEANINGLESS REGARDING THE SYSTEM BEHAVIOUR.

| | Initialization time[s] | | Synthesis time[s] / × realtime | |
|---|---|---|---|---|
| Experiment | Desire | Samsung | Desire | Samsung |
| 1 | 5.43 | 1.24 | 49.60 / 17.14 | 46.1 / 18.42 |
| 2 | 0.19 | 0.23 | 42.68 / 19.92 | 43.4 / 19.60 |
| 3 | 0.19 | 0.23 | 42.45 / 20.02 | 42.7 / 19.92 |
| 4 | 0.20 | 0.23 | 42.84 / 19.84 | 44.6 / 19.07 |
| 5 | 0.22 | 0.22 | 42.41 / 20.04 | 48.4 / 17.56 |
| Mean | 0.20 (median) | 0.23 (median) | 43.99 / 19.39 | 45.0 / 18.91 |
| Realtime: | 849.90 | | | |

the Table that the first initialization took approximately 5.5 seconds on HTC, which is not negligible, yet still acceptable, amount of time. The Samsung was significantly faster in reading the inventory from storage memory, which may be caused by the different technologies used for storage memory — we have checked by *AnTutu* benchmark that Samsung reads data about 4times(!) faster from the storage.

The following initializations took only a few dozen milliseconds, most likely due to data caching, where the database is read from system's file cache in memory, instead of from the much slower storage memory. The battery exhaustion during the initialization was unmeasurable. Also the performance of the synthesis is rather good and comparable on both devices – synthesizing text almost 20 times faster than it is played guarantees enough power reserve for other concurrently running applications and the system itself. Such numbers, however, are not very surprising, considering the intended simplicity of methods used in the SUI version. The battery exhaustion was unmeasurable for each individual experiment, with about 2% of energy consumed by HTC during the whole experiment (it took about 4minutes of pure TTS runtime, but 70minutes of speech were generated), with the device's display switched on. Samsung consumed less that 1% of battery power during synthesis (although we were not able to get exact number). It is not surprising, regarding its larger battery capacity.

### C. ARTIC MUI Version

The following results are related to the MUI version. Although this version is rather intended for desktop computers or servers, it is very interesting to look how it performs on the lower-resource device. For the experiment, however, there had to be a few adjustments of the system used otherwise. Firstly, aggressive pruning [17] was switched on. This speeds up the selection algorithm more than 50 times, without a considerable impact on speech quality observed [17]. Secondly, the size of speech segment database was shrank from 590MB to about 360MB. It has been achieved by a technique which preserves phrases containing only the most often used candidates [13], with coverage set to 85% (phrase was transformed into database only if it contains any candidate out of those 85% candidates used most frequently to create synthetic speech). Such coverage has been chosen ad-hoc, making speech database small enough to be mapped to the smartphone's memory and preserving a rather larger number of candidates. Similar to SUI version, further size reduction can be achieved by lowering the coverage, which, however, may have a negative impact on the quality of synthetic speech since fewer unit candidates are available in the selection process. Except those two adjustments, there were no other changes involved either in TTS engine or speech segments database.

The results of the experiment are shown in Table III. The initialization part of the TTS engine was unacceptably

TABLE III
THE PERFORMANCE OF ARTIC MUI INITIALIZATION AND SYNTHESIS.

| | Initialization time[s] | | Synthesis time[s] / × realtime | |
|---|---|---|---|---|
| Experiment | Desire | Samsung | Desire | Samsung |
| 1 | 177.65 | 50.8 | 137.43 / 6.12 | 257.35 / 3.27 |
| 2 | 191.61 | 56.8 | 135.80 / 6.20 | 182.25 / 4.62 |
| 3 | 188.59 | 60.6 | 137.96 / 6.10 | 253.47 / 3.32 |
| 4 | 195.11 | 57.2 | 135.05 / 6.23 | 196,90 / 4.27 |
| 5 | 189.92 | 53.9 | 143.43 / 5.87 | 169.45 / 4.97 |
| Mean: | 188.58 | 55.9 | 137.94 / 6.10 | 211.89 / 4.09 |
| Realtime: | 841.50 | | | |

ISBN: 978-988-18210-9-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

slow on HTC, taking more than 3 minutes to get the engine ready. However, considering that 360MB must be read from SD card and processed virtually Byte-by-Byte, it is not surprising. Moreover, there is also no "caching effect" observable, again, most likely due to the size of the segment database. About 3–4% of battery power were consumed for all the 5 consecutive experiments, with display switched on. Samsung device was faster again, with unmeasurable battery exhausting (which is expected since the speed of reading). Still, 50 seconds are rather slow for everyday use, as the application may be removed from memory any-time by the OS. On the other hand, the performance of synthesis looks fairly promising, especially for HTC. It is able to generate speech more than 6 times faster than it can be played, with about 4% of energy consumption for both devices (for 11 minutes of pure TTS runtime, with display being on as well) – such behaviour could already be acceptable. Samsung was slower in speech synthesis, most likely due to its slower FPU (checked also by benchmark).

Surely, the lower the inventory size, the better performance can be expected. The unclear issue pertaining to the reduction, carried out by lowering the number of unit candidates, is, however, the quality of synthesized speech. It was not examined during the experiment due to being unrelated to the topic of the paper.

## IV. Conclusion

It was shown that the ARTIC can run fairly well on devices powered by Android OS. It encourages us that it can be transferred (without significant effort) onto any of today's platforms supporting at least a basic set of standard C++ features.

It is also clear that the SUI approach should still be considered for its simplicity and small runtime requirements, even contrary to HMM synthesis with much more complicated signal decoder. Although it can be objected that the SUI-generated speech sounds noticeably less natural than that MUI-generated, the more background noise surrounds the listener (i.e. on street, in shop, or when driving a car), the less the difference is being focused on by listeners.

As regards the MUI version, mostly its initialization performance suffers on the devices with slower storage memory access. To some extend it can be overcome by a better signal compression methods — the examples presented were compressed about 4 times when compared to storing full waveforms of speech units. We are close to finishing new method achieving approximately 9 times compression, which is expected to lead to 240 MB for full inventory or 160 MB for the reduced inventory respectively. Moreover, a different candidates reduction technique (e.g. [12]) can be used to exclude rarely used unit candidates from the units inventory more effectively. The impact of these methods on the quality of synthetic speech, however, still needs to be addressed.

Regarding the planned application, introducing the need of the transformation of ARTIC to platforms with limited resources, it was shown that to employ ARTIC as the TTS backend will not be virtually any obstacle.

## References

[1] http://en.wikipedia.org/wiki/Android_(operating_system) (online, accessed during June 2011).

[2] J. Matoušek, D. Tihelka, and J. Romportl, "Current state of Czech text-to-speech system ARTIC," in *Text, Speech and Dialogue*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, vol. 4188, pp. 439–446.

[3] Z. Krňoul, M. Železný, "Realistic face animation for a Czech Talking Head," in *Text, Speech and Dialogue*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, vol. 3206, pp. 603–610.

[4] Z. Krňoul, J. Kanis, M. Železný, L. Mller, "Czech Text-to-Sign Speech Synthesizer," in *Text, Speech and Dialogue*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, vol. 4892, pp. 180–191.

[5] J. Matoušek, D. Tihelka, J. Psutka, "Experiments with automatic segmentation for Czech speech synthesis," in *Text, Speech and Dialogue*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, vol. 2807, pp. 287–294.

[6] J. Matoušek, J. Romportl, "Automatic Pitch-Synchronous Phonetic Segmentation," in Proceedings of $9^{\text{th}}$ Annual Conference of the International Speech Communication Association INTERSPEECH 2008, Brisbane, Australia, 2008, pp. 1626–1629.

[7] J. Matoušek, "On Minimizing the Size of Speech Unit Database in Concatenative Speech Synthesis," in Proceedings of $16^{\text{th}}$ Czech–German Workshop on Speech Processing, Prague, CR, 2006, pp. 70–76.

[8] http://www.comscore.com/Press_Events/Press_Releases/2011/3/comScore_Reports_January_2011_U.S._Mobile_Subscriber_Market_Share (online, accessed during June 2011).

[9] J. Romportl,J. Matoušek, D. Tihelka,"Advanced Prosody Modeling". in *Text, Speech and Dialogue*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, vol. 3206, pp. 441–447.

[10] D. Tihelka, J. Matoušek, "Revealing the most Significant Deterioration Factors in Single Candidate Synthetic Speech," in Proceedings of $10^{\text{th}}$ International Conference on Speech and Computer – SPECOM 2005, Patras, Greece, 2005, pp. 171–174.

[11] D. Tihelka, J. Matoušek, "Unit selection and its relation to symbolic prosody: a new approach," in Proceedings of $9^{\text{th}}$ International Conference on Spoken Language Processing Interspeech 2006 – ICSLP, Pittsburgh, USA, 2006, pp. 2042–2045.

[12] D. Tihelka, "Corpus-based Approach to Unit Selection Speech Unit Inventory Reduction in ARTIC TTS," in Proceedings of $17^{\text{th}}$ Czech–German Workshop on Speech Processing, Prague, CR, 2007, pp. 160–167.

[13] J. Matoušek, D. Tihelka, Z. Hanzlíček, "Reducing Footprint of Unit Selection TTS System by Excluding Utterances from Source Speech Corpus," in Proceedings of $19^{\text{th}}$ Czech–German Workshop on Speech Processing, Prague, CR, 2009, pp. 92–98.

[14] http://www.crystax.net/android/ndk.php (online, accessed during June 2011).

[15] J. Matoušek, J. Romportl, "Recording and Annotation of Speech Corpus for Czech Unit Selection Speech Synthesis," in *Text, Speech and Dialogue*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, vol. 4629, pp. 326–333.

[16] J. Matoušek, D. Tihelka, J. Romportl, "Building of a Speech Corpus Optimised for Unit Selection TTS Synthesis." in Proceedings of $6^{\text{th}}$ International Conference on Language Resources and Evaluation – LREC 2008, Marrakesh, Morocco 2008.

[17] D. Tihelka, J. Kala, J. Matoušek, "Enhancements of Viterbi Search for Fast Unit Selection Synthesis," in Proceedings of $11^{\text{th}}$ Annual Conference of the International Speech Communication Association – Interspeech 2010, Makuhari, Japan, 2010, pp. 174–177.

ISBN: 978-988-18210-9-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)