Exploring Web-Browser based Runtimes Engines for Creating Ubiquitous Speech Interfaces

Paul R. Dixon[1], *Sadaoki Furui*[2]

[1]National Institute of Information and Communications Technology, Japan
[2]Department of Computer Science, Tokyo Institute of Technology, Japan
paul.dixon@nict.go.jp, furui@furui.cs.titech.ac.jp

Abstract

This paper describes an investigation into current browser based runtimes including Adobe's Flash and Microsoft's Silverlight as platforms for delivering web based speech interfaces. The key difference here is the browser plugin is used to perform all the computation without any server side processing. The first application is an HMM based text-to-speech engine running in the Adobe Flash plugin. The second application is a WFST based large vocabulary speech recognition decoder written in C# running inside the Silverlight plugin.

Index Terms: speech synthesis, speech recognition, WFST

1. Introduction

When deploying a research-based speech system outside of the lab environment a problem often encountered is how to smoothly make the core technology available to other researchers, developers or directly to end users. Recently there has been a great deal of interest in cloud based solutions in which the speech processing engines are hosted in large data centres. Even though this approach has many advantages, there is the potential for very high running costs and it neglects the huge processing power that is distributed through the web.

In this paper we examine what is currently possible using rich-client based technologies. A key motivation was to maintain the simplicity of a web deployment model but offload all of the computation to the client. Even though the idea of web based speech interfaces has been investigated by others. We believe our approach is one of the first to investigate modern ubiquitous browser based runtimes and in the case of Automatic Speech Recognition (ASR) we found that on modern hardware it is possible to run an entire large vocabulary recognition system inside the browser. Prospective applications are speech enabling websites to allow better access for as many users as possible without relying on a third-party processing server. Another potential application area is for education use, currently evaluating and experimenting with research speech technology often requires the compilation of software and more complicated installations procedures which can exclude some users.

The contributions of this work are: Development of an extremely compact Hidden Markov synthesis (HTS) engine[16] running entirely in Flash[6]. This includes a description of the process we use to perform the conversion of an existing TTS engine. Development of a new C# based large vocabulary recognition decoder executing in managed code on the .NET framework and extended to a proof-of-concept browser embedded (ASR) engine running within the Silverlight[9] runtime.

2. Browser Embedded Speech Synthesis

Adobe Flash[6] is a universal platform for applications. Flite+HTS Engine for Flash is an HTS based TTS engine running entirely in Flash. The idea is to make HTS run in any modern browser and allow dynamic text-to-speech to be added to any webpage with the entire synthesis performed on the client. The remainder of this section describes the process used to convert an existing C based TTS engine into a component running in a Flash applet.

Recently Adobe released a research compiler known as Alchemy [7] that can compile C/C++ to byte-code that executes inside the flash virtual machine. To rapidly develop a browser based speech synthesis instead of re-writing an entire system for Flash using Action Script, we took the existing the HTS_Engine and Flite codebases and compiled them to a Flash component (swc) by using the Alchemy compiler[7]. Because of the highly portable nature of the code no modifications required other than a small amount of shim code to expose the TTS functionality to the main Flash application which was written in Action Script. The choice of HTS was motivated by the extremely small size of the associated engine resource and models. After embedding the necessary model parameters files into the Flash applet the final size was only 1.8MB. No compression or alteration of the models was made so further size reductions should be possible.

This approach for converting applications to Flash should be relevant for converting existing C based tools and systems for use on the web, furthermore the entire system was built using freely available tools.

3. Browser Embedded Speech Recognition

Very recently various high performance Distributed Speech Recognition (DSR) systems have emerged to target different platforms and scenarios. Some systems target the recent proliferation of connected mobile devices such as the iPhone and Android-based smartphones. Another class of DSR services provides web-based recognition engines, for example *webASR*[12] is a web-based transcription service in which users upload data and can later receive transcriptions. The WAMI system[11] is targeted more at application developers and provides a plugin and JavaScript interface for creating browser-based applications that can use an Automatic Speech Recognition (ASR) service. A feature all of these systems have in common is that there are server side engines where the user sends the speech data and the recognition results are sent back from. These DSR-based approaches allow for powerful server side recognition systems to be used, however, the drawbacks of providing such a service are: it potentially requires a large amount of computational infrastructure; it consumes valuable

research and development time to construct and administer the infrastructure and security; and could have potential data protection issues.

In this paper we present a different idea and implementation for a web-based ASR system. The novel difference is we run the distribution in the opposite direction of a traditional DSR system and instead send the entire speech recognition engine with the models down to the client. The recognition is then performed locally on the client machine allowing low latency recognition without requiring any further server side resources. The core of the system is a new decoder called $T4$ which is written with multi-platform support and flexibility in mind.

4. T4 Decoder

One of the main goals of the T4 decoder is to have a high performance speech recognition decoder written in C# that is capable of running on various implementations of the .NET framework[8]. In addition to providing multi-platform support, it is hoped the portability will allow for the creation of new and different types of speech driven applications.

4.1. .NET Framework

The .NET framework is a runtime environment which provides functionality such as memory management and garbage collection, in addition to an extremely rich class library providing features including networking, threading and XML web services[8]. Variants of the .NET framework run on Windows, PocketPC, Zune media players, Microsoft's Azure cloud service and even the Xbox games console. Because the code is first compiled to byte code and then Just In Time (JIT) compiled by the .NET framework, it makes it possible to create a single cross platform binary. Open source implementations such as Mono allow the same .NET programs to also run on *nix and Mac platforms. Another advantage of building on top of the .NET framework is that the extremely rich base library is particularly useful for creating rich multiplatform applications. However, these features and flexibility come with certain runtime costs and in this work we also evaluate the suitability of the .NET framework as a runtime for a high performance speech recognition engine.

4.2. T4 for Silverlight

Silverlight is a cross-platform browser plugin that features a smaller version of the .NET framework along with rich multimedia features[9]. The aims are to harness Silverlight to combine a research-based Weighted Finite State Transducer (WFST) speech recognition system with the smooth web deployment model and ease of use for end users and application developers.

The core of the system is the T4 decoder which uses a Viterbi beam search algorithm on a pre-compiled WFST search network[15]. The decoder has both search and signal processing components and under Silverlight 4.0 support direct microphone input.

The construction and deployment of a web-based T4 system is as shown in figure 1: In the first stage the language model, pronunciation dictionary and HMM definitions are used to precompile an integrated WFST[15]. The WFST is combined with the acoustic model(AM) and other resources into a single integrated *PAK* file.

The deployment step is the most simple step, the engine and the PAK file(s) are placed with the website contents on a web-

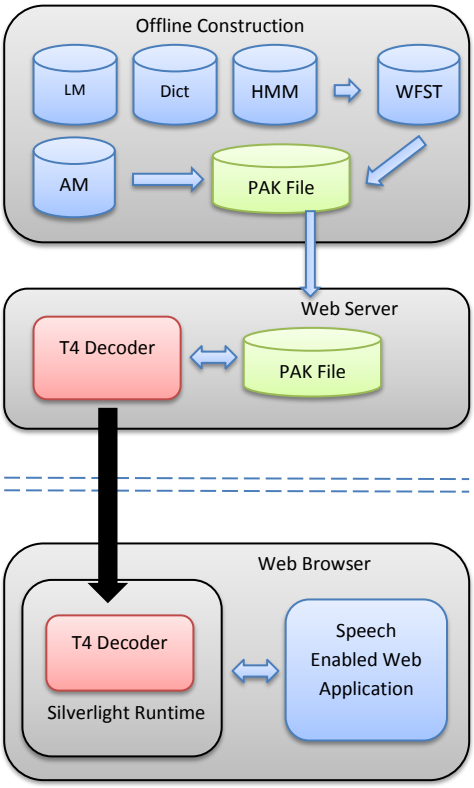

Figure 1: Diagram showing the model preparation and server-client architecture of the Silverlight decoder.

server. A huge advantage of our approach is that the recognizer will run locally on the client machine. Therefore, it doesn't require any further complicated server logic for the speech processing, just the ability to serve static files.

In the final stage when a client accesses the site with a supported browser, the engine is downloaded. The engine can then fetch the required models and make speech recognition available to the webpage for creating rich multi-modal applications. The engine can be used directly in either a rich Silverlight application or controlled via JavaScript by a standard browser application. Because the engine is running locally in the browser it allows for low latency recognition and thus permit classes of interesting speech recognition application, such as *audio joysticks*[3, 13]. It could also provide highly webpage specific recognition services which may be particularly beneficial for disabled users.

4.3. Memory Compression

To reduce the size for web deployment a simple yet more efficient memory representation for the WFSTs is used. The first size saving comes by discarding state information and only storing the arc information. This is achieved by encoding a *last arc* marker into the final arc of each state. To reduce the size of each arc a combination of a simple compression scheme known as *sub-bit precision*[4] packing with weight quantization is used. The arc input and output labels and the destination states can be rapidly packed using multiplication and addition operations and unpacked using division and remainder operations. After packing the arc labels and state information the remainder of the container space is used to store an index to a vector quan-

tized weight. The advantage of this scheme is we do not waste any sub-bits and the weight quantization has more flexibility as we do not need to round up (or down) to the nearest whole bit when constructing the quantization table.

The scheme is simple to implement and can yield good space saving. For example a compiled WFST with 15M arcs and 30M arcs was reduced from a disk image size of 630MB (in AT&T binary compiled format) to 230MB by packing each arc into 8 bytes, at this compression level the weight quantization did not degrade the recognition accuracy. This scheme can give comparable or better compression to the variable length scheme described in [5]. Here the authors reported similar compressed sizes for comparable sized acceptors (in terms of arcs and states) whereas we are using transducers which have an additional output label.

4.4. Multicore Acoustic Scoring

The acoustic scoring is often the largest CPU consumer during the decoding of clean speech and profiling the T4 decoder revealed the majority of CPU time is also spent on this computation. One particular speed-up technique that was found to be effective was to use the standard .NET `ThreadPool` to compute the required state scores for each frame in parallel. This technique involved creating batches of states and submitting them to the `ThreadPool`, although the latest version of the .NET framework support parallel extensions that allow for loops to be parallelized very easily. The `ThreadPool` has greater support across older versions of the .NET framework and is available in Silverlight.

5. Evaluation

The evaluation task was a large vocabulary speech recognition task evaluated on the Corpus of Spontaneous Japanese [14]. The test set used for the evaluation was composed of segmented utterances take from 10 lectures. The speech waveforms were first converted to sequences of 39 dimensional feature vectors with 10 ms frame rate and 25 ms window size. Each feature vector was composed of 12 MFCCs with deltas and delta-deltas, augmented with log energy, log delta and log delta-delta energy terms. The acoustic models were three-state left-to-right HMM tri-phone models and the complexity of GMM state models 39.

A Katz smoothed tri-gram language model G with a vocabulary size of 65k words was constructed using the AT&T GRM toolkit [2]. The $C \circ L \circ G$ search network was composed and optimized using the `dmake` tool from the AT&T DCD toolkit [1]. The final search network contained approximately 1.2M states and 2.4M arcs.

The T4 decoders were evaluated on Windows using .NET framework and Silverlight 4.0. The T^3 decoder was used as a baseline comparison [10]. In all cases the decoders could reach the same asymptotic word accuracy of 81.34%.

6. Conclusions

In this paper we have described the development of a small browser embedded TTS engine and also presented the initial implementation of a WFST speech decoder written in C# for the .NET framework that can run in the browser. Our investigations have not only shown that it is feasible to run a modern WFST decoder on the .NET framework, but we have also constructed a prototype system using the Silverlight runtime that makes it possible to deploy and execute an entire recognition system over the web using Silverlight. It is hoped that the prototype can be extended to build many different types of interesting distributed speech applications; one particular area we have in mind is to construct a large ad-hoc speech recognition grid for batch processing large amounts of data.

In addition to increasing the base feature set and improving performance, we plan to investigate ways to compress the acoustic models to allow for faster downloading. The addition of on-the-fly composition in conjunction with the ability to compile dynamic grammars on the client would be extremly beneficial in creating even more task specific flexibility for end users and creating an even more compact engine

Demo TTS and ASR systems are online at `http://www.furui.cs.titech.ac.jp/~dixonp/hts/index.html` and `http://www.furui.cs.titech.ac.jp/~dixonp/t4.html`.

7. References

[1] C. Allauzen, M. Mohri, M. Riley, and B. Roark. A generalized construction of integrated speech recognition transducers. In *Proc ICASSP*, pages 761–764, 2004.

[2] C. Allauzen, M Mohri, and B. Roark. Generalized algorithms for constructing statistical language models. In *Proc. of 41st Annual Meeting of the Assocication for Computational Linguistics*, pages 40–47, 2003.

[3] J. Bilmes, J. Malkin, X. Li, S. Harada, K. Kilanski, K. Kirchhoff, R. Wright, A. Subramanya, J. Landay, P. Dowden, and H. Chizeck. The Vocal Joystick. In *Proc. IEEE ICASSP*, May 2006.

[4] Jonathan Blow. Packing integers. `http://number-none.com/product/Packing\%20Integers/index.html`.

[5] Diamantino Caseiro and Isabel Trancoso. Using dynamic WFST composition for recognizing broadcast news. In *Proc. ICSLP*, pages 1301–1304, 2002.

[6] Adobe Corporation. Adobe flash player. `http://www.adobe.com/products/flashplayer//`.

[7] Adobe Corporation. Alchemy. `http://labs.adobe.com/technologies/alchemy/`.

[8] Microsoft Corporation. Overview of the .NET framework. `http://msdn.microsoft.com/en-gb/library/a4t23ktk.aspx`.

[9] Microsoft Corporation. Silverlight overview. `http://msdn.microsoft.com/en-us/bb187358.aspx`.

[10] P. R. Dixon, D. A. Caseiro, T. Oonishi, and S. Furui. The Titech large vocabulary WFST speech recognition system. In *Proc. ASRU*, pages 1301–1304, 2007.

[11] A. Gruenstein, I. McGraw, and I. Badr. The WAMI toolkit for developing, deploying, and evaluating web-accessible multimodal interfaces. In *Proc. IMCI*, pages 141–148, 2008.

[12] T. Hain, A. El Hannani, S.N. Wrigley, and V. Wan. Automatic speech recognition for scientific purposes – webASR. In *Proc. ICSLP*, pages 22–26, 2008.

[13] T. Kawasaki, T. Oonishi, and S. Furui. Voice-based direct manipulation for 3D interface. In *Proc. Interaction*, Januray 2009.

[14] K. Maekawa. Corpus of spontaneous Japanese: Its design and evaluation. In *Proc. ISCA and IEEE Workshop on Spontaneous Speech Processing and Recognition*, pages 7–12, 2003.

[15] M. Mohri, F. C. N Pereira, and M. Riley. Speech recognition with weighted finite-state transducers. *Springer Handbook of Speech Processing*, pages 1–31, 2008.

[16] Heiga Zen, Keiichiro Oura, Takashi Nose, Junichi Yamagishi, Shinji Sako, Tomoki Toda, Takashi Masuko, Alan W. Black, and Keiichi Tokuda. Recent development of the HMM-based speech synthesis system (HTS). In *In Proc. APSIPA*, 2009.