A Scalable Architecture For Web Deployment of Spoken Dialogue Systems

Matthew Fuchs[1], **Nikos Tsourakis**[2], **Manny Rayner**[2]

[1] Paideia Inc, 2225 E. Bayshore Rd, Suite 200, Palo Alto, CA 94303, California
[2] University of Geneva, TIM/ISSCO, 40 bvd du Pont-d'Arve, CH-1211 Geneva 4, Switzerland
mattfuchs@paideiacomputing.com, {Nikolaos.Tsourakis,Emmanuel.Rayner}@unige.ch

Abstract

We describe a scalable architecture, particularly well-suited to cloud-based computing, which can be used for Web-deployment of spoken dialogue systems. In common with similar platforms, like WAMI and the Nuance Mobile Developer Platform, we use a client/server approach in which speech recognition is carried out on the server side; our architecture, however, differs from these systems in offering considerably more elaborate server-side functionality, based on large-scale grammar-based language processing and generic dialogue management. We describe two substantial applications, built using our framework, which we argue would have been hard to construct in WAMI or NMDP. Finally, we present a series of evaluations carried out using CALL-SLT, a speech translation game, where we contrast performance in Web and desktop versions. Task Error Rate in the Web version is only slightly inferior that in the desktop one, and the average additional latency is under half a second. The software is generally available for research purposes.

Keywords: Speech recognition, web, evaluation, CALL

1. Introduction and motivation

Since the mid-90s, both Web and speech technology have made huge strides, but they have only recently started to come together in earnest. Although the advantages of deploying spoken dialogue systems on the Web hardly require elaboration, the technical problems involved have been surprisingly hard to overcome.

In this paper, we describe a scalable architecture, particularly well-suited to cloud-based computing, which can be used for Web-deployment of spoken dialogue systems. In common with similar platforms, like WAMI ((Gruenstein et al., 2008); `http://wami.csail.mit.edu`) and the Nuance Mobile Developer Platform (NMDP; `http://dragonmobile.nuancemobiledeveloper.com`), we use a client/server approach in which speech recognition is carried out on the server side; our architecture, however, goes further than these systems by performing dialogue management, application integration, and large-scale grammar-based language processing in the cloud, rather than just returning the results of recognition to the client. Another important difference is that speech is passed to the recognition processes in the form of files, rather than using streaming audio. Although this goes against the currently prevailing wisdom, we have found that there are compensating advantages, and that the performance hit, with a little care, can be reduced to an acceptable level.

Our exposition will be based on two large applications built using our framework, which we argue would have been hard to construct in WAMI or NMDP. Although both applications are speech-enabled games, they differ in several respects, both in terms of functionality and architecture. Minion Dominion (Chua and Rayner, 2010), the first one built, was commercial in nature, and was deployed as part of the marketing for the Universal Studios movie *Despicable Me*; it allowed two animated characters to be controlled by spoken or typed commands, using a vocabulary of about 400 words. The game was visited by over one million people, and at its peak, shortly after release of the movie in July 2010, was getting more than 50K hits a day. The game was developed together with a third party who was responsible for the animation aspects, and the speech and dialogue processing modules were accessed through a Web client deployed by the third party.

CALL-SLT ((Rayner et al., 2010); `http://callslt.org`) is in contrast a language-learning application developed at Geneva University, which implements a version of the "translation game" idea from (Wang and Seneff, 2007): the system prompts students with abstract representations of what they are supposed to say, and uses speech translation technology to compare the result with the correct answer. This system uses a Flash-based client, also developed by Geneva University, which connects directly to the other components.

We used CALL-SLT to perform a series of evaluations, where we contrasted performance in Web and desktop versions. Task Error Rate in the Web version was only slightly inferior to that in the desktop one, and the average additional latency was under half a second.

The rest of the paper is organised as follows. Section 2. describes the architecture and Section 3. the evaluation. Section 4. concludes.

2. Architecture

This section presents an overview of the architectural framework. Applications, including in particular the two we focus on here, contain four core types of runtime components: a Speech Router, one or more Dialogue Servers, one or more Recognition Servers, and several Clients. This is designed to be extensible; as we will see, adding additional components, such as TTS servers, is straightforward.

The first two types of components are both application-independent and run on the server machines. The Speech Router is the top-level process; it handles the connection to the internet and mediates message traffic between the Clients, the Dialogue Servers and the Recognition Servers.

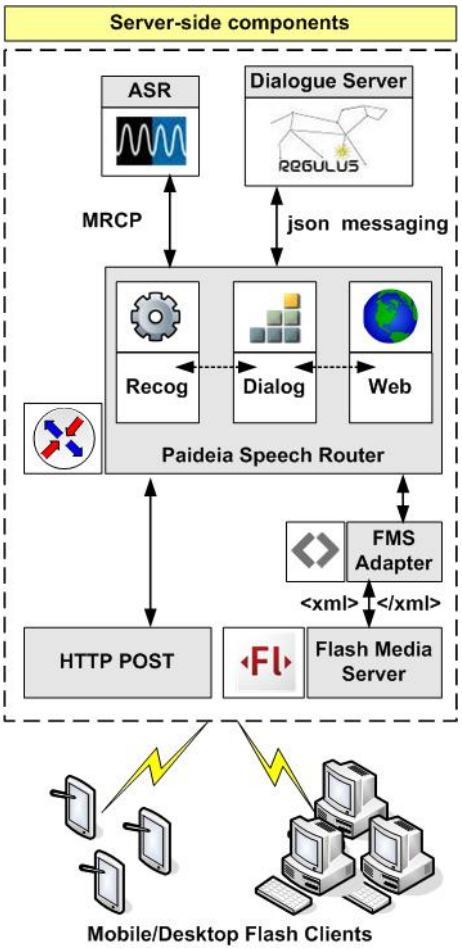

Figure 1: Top-level architecture of the framework (configuration used for Minion Dominion).

The Dialogue Servers perform language analysis and dialogue processing. The Recognition Servers are third-party software, and also run on the server machines; so far, we have used the Nuance Toolkit, but any recognition engine capable of supporting the MRCP protocol would work equally well. The Client is an application-specific process running on the user's internet-enabled device.

As already noted, the top-level configurations of the two sample applications are slightly different. Figures 1 and 2 give a schematic pictures of the message flow between components in each case. We now describe both the components and the message flow in more detail.

2.1. The Speech Router

The Speech Router is an embedded web server designed to connect the Clients to cloud-based recognition and applications. It is the front end of a multi-tiered architecture permitting any number of Clients to talk to any number of applications with any number of languages and grammars; if load considerations require multiple Recognition Servers and Dialogue Managers, the Speech Router is responsible for passing speech files and recognition results between them, performing associated load balancing.

We will first discuss the basic flow of messages in the system and then describe the changes to support Flash. Note

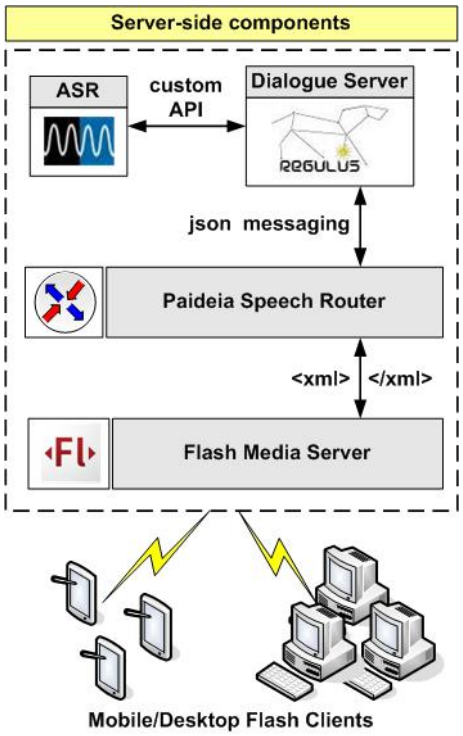

Figure 2: Top-level architecture of the framework (configuration used for CALL-SLT).

that we describe the passage of messages through this system in a fairly bare-bones manner — more complex applications will have more involved session start up and tear down, but that doesn't affect the basic flow. The responsibilities of the different components are discussed here:

1. The Speech Router embedded web server receives an HTTP POST message from a client. The message, at a minimum, must identify the application it is for. If it is for a new session, then a session ID is created, otherwise a session ID will be included. There may other parameters attached, such as a voice file or other user actions. We have adopted JSON as the format for all non-speech data. With more sophisticated applications and clients, there may be user information in a database or memcached to use for further processing. Finally, a message ID is generated.

   Once it has enough information, the Speech Router specifies a processing pipeline for the message. For a spoken message this would include a Recognition Server, a Dialogue Server, and the address of the current Speech Router instance — the response must eventually arrive back at the same front end to respond to the remote client. Without a speech payload, the pipeline will go directly to the Dialogue Server and then back to the front end. These choices are based on configuration files specifying host and process IDs for all recognisers and applications. Currently all processes communicate via HTTP. We plan to move configuration information into memcached and replace HTTP with reliable queues to support greater reliability and flexibility.

If there is a speech payload and it is not in the required format for the recogniser, (for example, MP3 instead of WAV), the front end performs the relevant conversion before sending the message to the next layer.

2. At the Recognition Server there is a Java process which performs the following:
   (a) receive the message,
   (b) store the speech file,
   (c) determine the grammar to be used,
   (d) communicate with the recogniser using MRCP,
   (e) when it receives the recognition response, forward that to the next process in the chain — usually the Dialogue Server if there are no errors
3. The Dialogue Server receives its input either from the Recognition Server, if there was speech, or from the Speech Router. It uses the session ID to retrieve the state, if any, for the session, processes it, and sends the results along to the next process in the chain, which is usually back to the original Speech Router process. The Dialogue Server consists of a Java process that communicates with the rest of the system and the Regulus server (see below).
4. At the end, the message returns to the Speech Router. The results are passed to the servlet thread handling this communication, which logs some performance data and then returns the results from the Dialogue Server to the client in the response to the initial POST.

This architecture is designed to support the straightforward addition of more levels. For example, we did not require TTS for the Despicable Me application, but it could easily have been added by inserting a TTS process into the chain after the Dialogue Sever and before the results return to the Speech Router. Other application components could likewise be inserted along the way.
Flash is handled, as usual, through another layer of indirection. A small Java process sits between the Flash Media Server and the front end described above. This layer communicates with FMS using XML messaging, cracks open Adobe's proprietary FLV format to retrieve the embedded mp3 speech file, and then communicates with the server like any other HTTP POST client.

2.2. The Dialogue Server

The Dialogue Server is a process implemented on top of the Open Source Regulus Platform (Rayner et al., 2006), which encapsulates the platform's runtime functionality in a way that makes it easy to integrate into a distributed application. At an abstract level, the Regulus dialogue management framework (Rayner et al., 2006, Chapter 6) implements a version of Update Semantics (Larsson and Traum, 2000). The central concepts are those of *dialogue move*, *information state* and *dialogue action*. At the beginning of each turn, the dialogue manager is conceptually in an information state. Inputs to the dialogue manager are by definition dialogue moves, and outputs are dialogue actions. The

Figure 3: CALL-SLT application running on the Samsung Galaxy Tab.

behaviour of the dialogue manager over a turn is completely specified by an *update function* f of the form

$$f : State \times Move \rightarrow State \times Action$$

Thus if a dialogue move is applied in a given information state, the result is a new information state and a dialogue action. The important point here is that dialogue processing is completely side-effect free: behaviour is determined just by the state and the dialogue move. This means that the dialogue server can easily maintain multiple parallel dialogue sessions, associating each one to a separate state object, which is saved in a database between turns under a unique session ID.

The Dialogue Server communicates with other processes using JSON messages sent over a socket. The use of Unicode-based JSON makes it easy to support speech and language processing for languages with non-ASCII character sets. The CALL-SLT application, in particular, has versions which require handling of text in Japanese, Chinese, Arabic and Greek.

2.3. Recognition Servers

Recognition for our two applications uses Nuance 9.0 for Despicable Me and Nuance 8.5 for CALL-SLT; the current modules are able to communicate with any recogniser platform capable of handling MRCP and either GrXML or GSL. It would not in fact be difficult to allow use of other platforms, which would only require rewriting the current MRCP client to use some different protocol. We have in particular been considering extending the framework to support the Sphinx recogniser ((Lee et al., 1990); `http://cmusphinx.sourceforge.net/`). Here, we might link directly to the Java library.

2.4. Clients

Using the framework described earlier, it is straightforward to implement speech clients that follow uniform design principles. The developer is concerned with the following tasks: 1) creating the graphical user interface, 2) maintaining the application's state logic, 3) performing recording and playback of audio, and requesting services from the remote peer. All applications are developed using Flash 11 in combination with ActionScript 3.0. For the Android platform, where access to the microphone is, annoyingly, not available inside the browser, we had to create a standalone application using the Adobe AIR 2.6 runtime. Figure 3 shows a screenshot of the GUI for the mobile version of the CALLSLT system.

When the user types the application's link in the browser, the remote web server delivers the requested page that includes the ".swf" file of our flash client. The latter takes control and communicates directly using Remote Procedures Calls (RPC) with the remote Flash Media Server, which is the entry point of any client's request. As the Flash framework dictates, the end user must allow our application to access the microphone before it can start using it.

An important aspect of the GUI is the way the recognition button is used. Due to the limitations of the target platform (lack of an endpointing mechanism), we have adopted a push-and-hold solution, where the user has to keep the button pressed while speaking. The recorded audio packets are streamed to the server until the button is released. The latter signifies the end of the user speech, triggering the transition to the next step of the processing chain; recognising using the remote audio file, processing the result and returning a response to the client.

3. Experiments

We carried out three simple experiments using the CALL-SLT application, which we now describe.

3.1. Recognition accuracy

The first experiment was designed to compare recognition accuracy in the Web-based and desktop versions of the app. We asked eight subjects to interact with the English-for-French-speakers version of CALL-SLT, running on desktop, Web and mobile platforms in a quiet office environment. The mobile version was run on a Samsung Galaxy tablet using the onboard microphone. All subjects were good but non-native speakers of English, and either native French speakers or strong second-language speakers. The subjects were asked to do 10 examples for each of three different lessons. Responses were typically about 5 to 7 words long for the first two lessons, and about 10 to 12 words long for the third one.

The results are summarised in Figure 4 (Word Error Rate) and 5 (Task Error Rate). It is apparent that average WER is better on the desktop than on the Web version (6.2% versus 7.5%), though TER was closer; two of the seven users in fact scored better TER on the Web version. The mobile version was marginally worse than the Web one.

The main factor responsible for these differences appears to be a divergence in user interface functionality: the desktop version is push-to-talk, while the Web and mobile versions are push-and-hold. It is clear that subjects find push-and-hold less user-friendly, a problem which has also been noted in the WAMI framework.

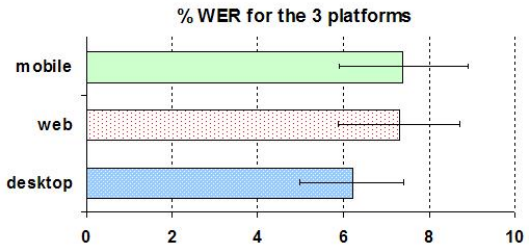

Figure 4: WER for desktop, Web and mobile versions of English CALL-SLT.

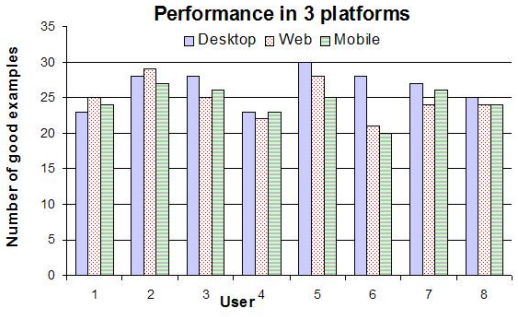

Figure 5: Task performance on desktop, Web and mobile versions of English CALL-SLT, out of 30 examples attempted, on 8 non-native speakers.

3.2. Server utilisation

The second experiment focussed on server utilisation. We took 10 session logs from an evaluation exercise (Bouillon et al., 2011) using the French-for-Chinese-speakers version. Session lengths varied from 6 minutes to an hour and a half, averaging 52 minutes. Figure 6 shows the results of replaying the logged server commands offline to estimate server load for the commands used in each session. Utilisation varies from 4% to 14%, depending mainly on the extent to which subjects interacted with the system, with a mean value of 7%. This agrees with the intuitive observation that the system is habitable with up to about ten con-

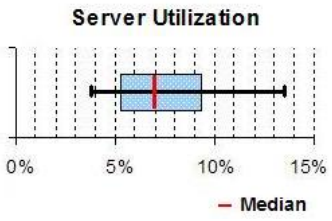

Figure 6: Box plot for server utilisation, expressed as a percentage, for 10 sample sessions using the French-Chinese version of CALL-SLT. The blue box is constructed to contain the 50% of sessions closest to the mean.

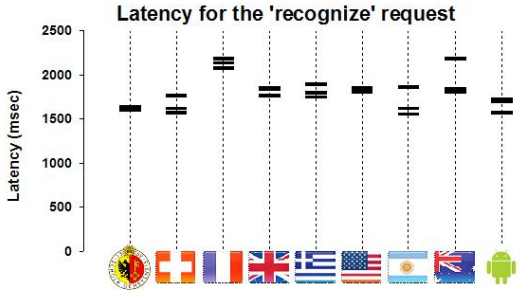

Figure 7: Average latency of "recognise" messages.

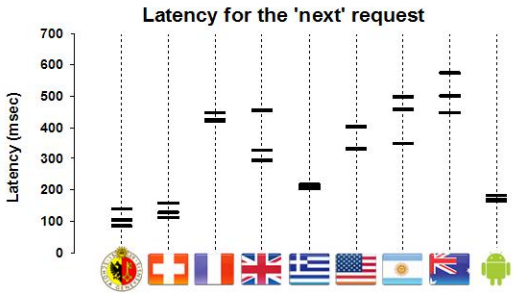

Figure 8: Average latency of "next prompt" messages.

current users.

3.3. Latency

In the third experiment, we implemented a Flash client which simulated running a standardised CALL-SLT session from a variety of remote locations and measured latency for the three most common types of message: "recognition and match", "next prompt" and "get help for current prompt". In each case, latency was defined as the time between the client's sending the message and receiving a reply. We used nine different locations, ranging in distance from inside the University building (leftmost column) to over 10 000 kilometres away, and ran the simulated session at three different times of day for each location, with times chosen to get varied levels of internet load. The rightmost column shows results for an Android device used on the University Wifi network.

Figures 7 to 9 summarise the results. There is only weak correlation with location. Latency varies from about 0.1 to 0.5 sec for "next", from about 0.3 to 1.0 sec for "help", and from about 1.5 to 2.0 sec for "recognition and match". Offline tests with recorded files showed that server processing times for 'recognition and match" averaged around 1.3 sec; this suggests that the extra recognition response latency resulting from the Web framework is about 0.2 to 0.7 sec. Since audio files average about 3 seconds in length, and recognition speed is about 0.08×RT, fully streaming audio would have reduced this by about 3000×0.08 = 240ms. The minor performance hit associated with our solution seems in practice entirely acceptable.

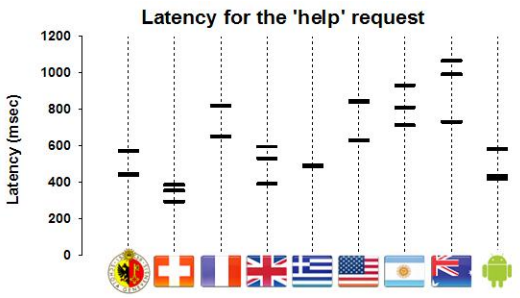

Figure 9: Average latency of "help" messages.

4. Summary and conclusions

We have described a framework for deploying speech-enabled dialogue applications over the web, and shown how it has been used to build two substantial systems of this kind. The framework has several distinguishing features which differentiate it from similar offerings, like WAMI or NMDP. In particular, it includes extensive support for recognition using large grammar based language models and server-side (cloud-based) dialogue management; it allows easy scalability; and it uses a file-based protocol to communicate with the recognition server, instead of using streaming audio throughout.

It is not obvious that either of the sample applications could easily have been constructed using WAMI or NMDP. Both apps, particularly CALL-SLT, require substantial medium-vocabulary grammars (hundreds of words and thousands of rules), which is incompatible with the other platforms' approaches to recognition: NMDP only offers a general large-vocabulary solution, and WAMI is not designed to be used with large recognition grammars. Both apps, particularly Minion Dominion, also needed to support a substantial number of parallel conversations. If we had used WAMI, the lack of support for scalability would have posed many problems.

Although fully streaming audio would have reduced recogniser latency a little, as discussed in Section 3.3., it would have been extremely expensive to acquire servers and licenses sufficient to support the large number of channels we required. Partially file-based processing gave us an acceptable solution at a low cost; since recognition speed was about 15 to 20 times real-time, a single channel could comfortably handle about ten parallel users per recognition server.

In fact, if we back off from the narrow issue of communication with the recognition server and consider the application as a whole, it is entirely possible that use of WAMI or NMDP would have resulted in *greater* latency. Both our apps include significant server-side application processing, and recognition results are immediately available for further processing in the cloud. The alternatives would require first returning recognition results to the client and then shipping them up to a server for dialogue management and other processing. These two additional hops over the public internet, perhaps using 3G, 4G, or wifi connections, are the slowest links in the system and would probably add more latency than file-based processing does. Anecdotally, when we have demonstrated CALL-SLT in public, we have usually received positive comments about the fast response

times.

The one place where we really felt that we could have won by using NMDP was in endpointing. NMDP contains support for client-side endpointing, something that our framework conspicuously lacks. The result is that we are forced to use a push-and-hold interface; as discussed in Section 3.1., users do not like this, and it impacts negatively on performance. We are currently investigating ways to integrate third-party endpointing solutions into the framework.

Availability of software

Paideia has made the speech router components available to the University of Geneva on a research-only license. For similar arrangements, contact the authors.

5. References

P. Bouillon, M. Rayner, N. Tsourakis, and Q. Zhang. 2011. A student-centered evaluation of a web-based spoken translation game. In *Proceedings of the SLaTE Workshop*, Venice, Italy.

C. Chua and M. Rayner. 2010. What's the Magic Word? In *Proceedings of the Thirteenth Australasian International Conference on Speech Science and Technology*, Melbourne, Australia.

A. Gruenstein, I. McGraw, and I. Badr. 2008. The WAMI toolkit for developing, deploying, and evaluating web-accessible multimodal interfaces. In *Proceedings of the 10th international conference on Multimodal interfaces*, pages 141–148. ACM.

S. Larsson and D. Traum. 2000. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering, Special Issue on Best Practice in Spoken Language Dialogue Systems Engineering*, pages 323–340.

K.F. Lee, H.W. Hon, and R. Reddy. 1990. An overview of the sphinx speech recognition system. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 38(1):35–45.

M. Rayner, B.A. Hockey, and P. Bouillon. 2006. *Putting Linguistics into Speech Recognition: The Regulus Grammar Compiler*. CSLI Press, Chicago.

M. Rayner, P. Bouillon, N. Tsourakis, J. Gerlach, M. Georgescul, Y. Nakao, and C. Baur. 2010. A multilingual CALL game based on speech translation. In *Proceedings of LREC 2010*, Valetta, Malta.

C. Wang and S. Seneff. 2007. Automatic assessment of student translations for foreign language tutoring. In *Proceedings of NAACL/HLT 2007*, Rochester, NY.