# Web-based automatic speech recognition service - webASR

*Stuart N. Wrigley, Thomas Hain*

Department of Computer Science, University of Sheffield
Regent Court, 211 Portobello, Sheffield S1 4DP, UK
`s.wrigley@dcs.shef.ac.uk, t.hain@dcs.shef.ac.uk`

## Abstract

A state-of-the-art automatic speech recognition (ASR) system was developed as part of the AMIDA project whose core domain was the transcription of small to medium sized meetings. The system has performed well in recent NIST evaluations (RT'07 and RT'09). This research-grade ASR system has now been made available as a free web service (webASR) targeting non-commercial researchers. Access to the service is via and standard browser-based interface as well as an API. The service provides the facility to upload audio recordings which are then processed by the ASR system to produce a word-level transcript. Such transcripts are available in a range of formats to suite different needs and technical expertise. The API allows the core webASR functionality to be integrated seamlessly into applications and services. Detailed descriptions of the system design and user interface are provided.

**Index Terms**: speech recognition, interface, api, transcription, web service

## 1. Introduction

Automatic speech recognition (ASR) is becoming a common feature of many products and applications ranging from smartphone utilities to automated call centres and beyond. However, the majority of such services tend to tailored to the specific needs of the customer; in this way, optimal performance can be gained by training and testing the system on a specific domain. The disadvantage of this approach is that (commercial) effort is largely focused on high return markets (professional dictation, etc); speech recognition systems for other fields is usually embedded in applications and not freely accessible.

Natural language processing applications are thriving and, similar to speech processing, attention is increasingly focussed on texts which were previously unavailable in machine readable form such as broadcast news and discussions. More recently transcripts of small to large group meetings such as tutorials, lectures, committee meetings, court-room proceedings, etc. have also seen increased interest. Current research is normally based on well annotated corpora such as the AMI corpus [1]. However, for many databases, speech transcripts are not included. Furthermore, most groups interested in natural language research do not have access to research-grade speech recognition software. Similarly, in other fields such as sociological or educational research, an increased demand for transcripts can be observed.

In this paper we present a web-based interface[1] to our state-of-the-art speech recognition systems [2, 3]. The aim of this interface is to provide the scientific research community with an interface to free speech transcription for domains and applications where the generation of such transcripts was not previously feasible. We briefly describe the underlying ASR system before describing the design of the webASR service and the the functionality it provides.

[1]http://www.webasr.org/

## 2. ASR system

WebASR speech recognition systems are derived from the AMI and AMIDA systems for meeting transcription [4, 2] but are not confined to such domains. Other systems are available; for example, the transcription of telephone speech [4] and broadcast news data. Most systems are the result of collaboration with other research groups, most notably within the AMI consortium[2]. One of the key aspects of the webASR systems is grid computing-based offline processing in multiple stages, i.e. initial recognition passes are used to adapt the existing acoustic and language models. Table 1 illustrates the significant gain from initial to final pass. However, as later stages require more CPU time per percentage gain (note smaller improvement in performance following second half of processing compared with after first half of processing time in Table 1), normally the number of processing stages is reduced, statically or even dynamically during processing time. Naturally the adaptation performance is also a function of the amount of speech available; performance may remain poor when only short segments of audio are uploaded.

| Pass | Close-talking | Far field |
|---|---|---|
| Initial | 41.3 | 44.2 |
| After ≈ ½ processing time | 28.3 | 36.3 |
| Final | 27.2 | 33.2 |

Table 1: % WER on the NIST 2009 RT evaluation test set for webASR systems

### 2.1. ASR processing

Depending on the input, systems include automatic segmentation, speaker clustering, acoustic filtering, or beam-forming on the front, and post-processing such as confidence score estimation or fluency filtering. Typically the output from several system stages is provided in the resulting transcript files.

Since the API-based interface allows XML metadata (input segmentation information, manual transcripts or lists of words that are likely to occur) to be associated with the audio, enhanced stages such as adaptation of word lists and language models become possible, or even non-recognition tasks such as, for example, audio-to-text alignment.

The flexibility of the ASR systems deployed on webASR is

[2]http://www.amiproject.org/

achieved by an implementation using the resource optimisation toolkit (ROTK). Here the ASR processes are implemented as key processing modules that can be arranged flexibly and dynamically. The exact arrangement of the modules itself is subject to optimisation. When input data is provided it is not linearly fed through the processing graph. Instead data is split into parts, independently for each module, and then submitted to a grid computing facility. Automatic generation of module/data dependency allows the split of the overall task into hundreds, if not thousands, of pieces. The result is system output obtained at much lower latency than the actual CPU run-time for the data. This structure also makes it easy to add new systems and to control throughput by efficient use of compute resources.

## 3. WebASR design

The goal of webASR is to provide free access to state-of-the-art speech recognition to as wide a community as possible while at the same time having as low an adoption overhead as possible. To this end, it was decided that the core means of interacting with the service would be via a standard web browser; this removed the necessity for platform specific development, requires no software be installed on the user's computer and allows the service to be accessed anywhere in the world while still having access to all previous uploads and transcripts. The webASR service is implemented as a Java Servlet based web application hosted using the Apache Tomcat open source servlet container[3].

The webASR web application uses a standard design pattern known as the *model-view-controller* pattern (also known as the *front controller*, or *Model 2*) and was implemented entirely in Java. A Servlet – a special type of Java class (conforming to the Java Servlet API) which allows it to respond to HTTP requests – acts as the controller, providing a centralised point of control for all page requests. The controller is responsible for delegating requests to the appropriate Java class which processes the incoming request and data and makes any necessary changes to the underlying model. The model encapsulates all the domain-specific information on which the application operates. Upon successful completion, the controller then redirects the user's browser to the new page. Here, the model (persistent storage mechanism) was implemented using a *MySQL* database and the view (the user interface / web pages) used *JavaServer Pages* (JSP).

Care was taken during the database schema design to enable maximum flexibility as well as reduce inadvertent data loss by users. For example, most objects stored in the database have a boolean 'deleted' flag associated with them. When users delete an object that they own (e.g., an upload) this flag is set to true as opposed to physically deleting the object in the database. The advantage of this approach is that recovering lost data becomes trivial (and hence requires less time from administrators).

Since it was necessary to perform a number of client-side analyses on the files chosen for upload by the user (audio type, file size, etc.) it was decided to use a Java Applet. The use of an applet rather than a standard web page also allowed the upload process to be obfuscated to reduce the opportunities for malicious use.

However, Java virtual machines run applets under a different security regime than standard applications: applets are automatically considered 'untrusted' in order to protect the user's computing resources from malicious or faulty applets. Such applets are prevented from accessing the user's hard disk and any files on it. To resolve this issue, the applet must be signed using a digital certificate. The certificate specifies who the applet is from and contains the digital signature of the 'certificate authority'. The certificate authority is a trusted organisation (usually well known, such as VeriSign) which vouches for the applet's trustworthiness. The applet used here is currently self-signed but future versions will be signed by a recognised authority.

[3]Note that Tomcat should not be confused with the Apache web server, which is a conventional HTTP web server

Figure 1: CAPTCHA example to prevent bot / non-human registrants.

Access to the system is restricted to registered (and manually approved) users each of which are assigned a specific level of authority (e.g., *administrator*, *public user*, etc.) and an upload quota (e.g., 1 GB over 30 days). The quotas can be imposed in a very flexible manner simply by specifying the maximum upload amount over a specified duration. Both of these features prevent use of the system which could be deemed malicious or beyond *fair use*. Furthermore, the use of upload quotas allows us to impose broad, high-level, usage limits on the service since it is hosted on a conventional University-based network without the resources for industrial strength load balancing and bandwidth provisions.

A large number of audio file formats exist and for each one there are a plethora of ways in which the audio samples can be encoded: sampling rate, bits per sample, number of channels and encoding algorithm (e.g, PCM, $\mu$-Law, etc.). Therefore, the webASR system keeps a detailed list of all the different audio formats which can be accepted for processing.

### 3.1. Browser interface

The primary mode of interaction with the service is via the browser-based interface.

#### *3.1.1. User functionality*

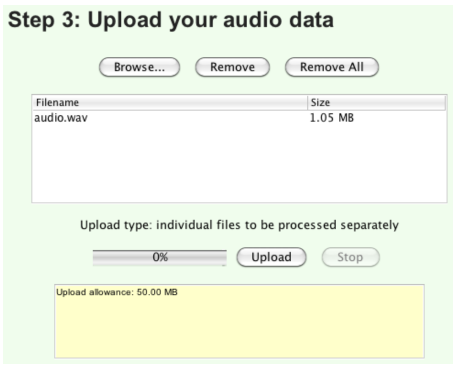
The first step to using the system is to register via the web form. This gathers basic information regarding the users name and affiliation. In addition, we employ a CAPTCHA test[4] which protects the service from unwanted registrations by bots. It generates and evaluates tests that humans can pass but current computer programs cannot. For example, only humans can read distorted text such as that shown in Fig. 1. The registration request is logged by the system and an appropriate message is displayed when a member of the 'administrator' group logs in.

Once approved (see Section 3.1.2), a user can perform three high-level activities: manage their profile, manage their existing uploads and, finally, upload audio files containing speech to be transcribed.

*Profile management.* This allows the user to ensure their login and contact details are up to date on the system.

*Audio upload.* Before ASR processing can proceed, the audio containing the speech must be uploaded together with relevant metadata. This metadata covers two broad areas: information about the environment in which the recording took place and information about the speech content itself. The former captures details of the number of microphones used, the type of micro-

[4]http://www.captcha.net/


Step 3: Upload your audio data

Browse... Remove Remove All

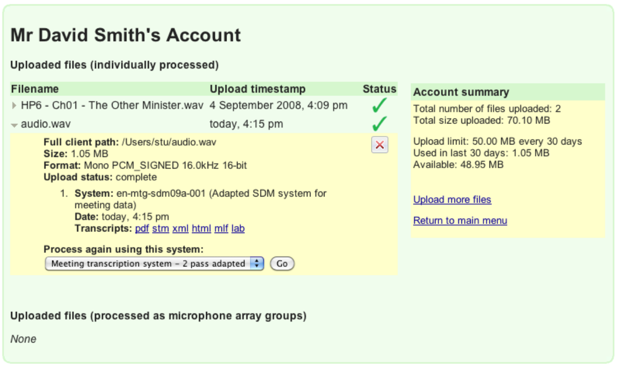
| Filename | Size |
|---|---|
| audio.wav | 1.05 MB |

Upload type: individual files to be processed separately

0% Upload Stop

Upload allowance: 50.00 MB

Figure 2: Java Applet to control upload of audio data.


Mr David Smith's Account

Uploaded files (individually processed)

| Filename | Upload timestamp | Status |
|---|---|---|
| HP6 - Ch01 - The Other Minister.wav | 4 September 2008, 4:09 pm | ✓ |
| audio.wav | today, 4:15 pm | ✓ |

**Full client path:** /Users/stu/audio.wav
**Size:** 1.05 MB
**Format:** Mono PCM_SIGNED 16.0kHz 16-bit
**Upload status:** complete

1. **System:** en-mtg-sdm09a-001 (Adapted SDM system for meeting data)
   **Date:** today, 4:15 pm
   **Transcripts:** pdf stm xml html mlf lab

**Process again using this system:**
Meeting transcription system – 2 pass adapted Go

Account summary
Total number of files uploaded: 2
Total size uploaded: 70.10 MB

Upload limit: 50.00 MB every 30 days
Used in last 30 days: 1.05 MB
Available: 48.95 MB

Upload more files

Return to main menu

Uploaded files (processed as microphone array groups)

*None*

Figure 3: Account information showing uploads and available transcripts.

phone (lapel, headset, farfield, array, etc.) and the physical location of the recording (street, office, etc.). By specifying more than one microphone, the user has the option of having them processed as a microphone array. In this case, additional beamforming processing is included to improve the speaker segmentation during the speech recognition processing. Metadata regarding the speech content covers aspects such as the number of participants, the gender mix, the type of conversation (discussion, free conversation, presentation, etc.) and the topics that appeared in the discourse.

The final stage is the selection of the audio files to be uploaded. This is achieved through the use of a Java Applet. As mentioned above, this approach was motivated by a number of factors. Firstly, it allows the system to inspect, *before upload*, each audio file selected by the user to ensure it is of a format supported by the system. Secondly it allows the system to ensure, again *before upload*, that any selected files would not exceed the user's upload allowance. Thirdly, if microphone array recordings have been specified in the metadata collection stage, it allows the system to ensure that the correct number of audio files (one per microphone) have been selected. Finally, if an upload fails for any reason, it can be resumed from the point of failure, thus minimising bandwidth wastage. None of these can be performed using a conventional HTML-based upload form. Once all the files have been selected (and the total size is within the user's upload quota), the files are transferred to the webASR servers for processing.

*Upload management.* Once one or more audio files have been uploaded, they, together with their associated ASR processing and transcripts, can be managed using the interface shown in Fig. 3. The user's account page provides information regarding the status of their account as well as details of all the uploads that they have made. For the example user shown in Fig. 3, the right hand panel provides information about their account: the number of files uploaded and the cumulative size of all their uploads over the whole lifetime of their account. In addition, information is provided regarding their upload quota: the size of uploads made during this quota period and the amount remaining.The lefthand side of the panel lists all the uploads ordered by date. Each upload has an icon associated with it to quickly convey the processing status of the audio upload. In this case, both audio uploads have successfully completed their ASR processing (indicated by a green tick). Other symbols indicate ongoing ASR processing or possible faults. For more information on each upload, the user is able to click on the grey arrow to the left of the filename to expose the *upload and transcript pane*. For each upload, aspects of the metadata are shown such as the path of the audio file (based on the user's filesystem not that of the webASR server), the audio file format and the file size. This information is useful to differentiate different uploads made in close succession.

The webASR service has been designed to allow each upload to be processed multiple times. This is particularly useful if the user has access to different ASR systems and wishes to experiment with those to, for example, determine which provides the optimal performance given a particular type of data. Each time an ASR system is used to produce a transcript of an audio file, the details of this processing is shown as a numbered list with the *upload and transcript pane*. In this example, the audio file has been processed once by the *en-mtg-sdm09a-001 (Adapted SDM system for meeting data)* system. Indeed, as can be seen from the drop down box at the bottom of the *upload and transcript pane*, the user has the option of having this audio file processed with an alternative ASR system; in this case one entitled *Meeting transcription system - 2 pass adapted*. For each ASR process, a transcript is generated which, internally, is stored as XML. This allows the service to offer a wide range of output formats (PDF, STM, XML, HTML, MLF, etc.) with maximum efficiency: all the available formats are created 'on-the-fly' using XSLT stylesheets applied to the base XML. This approach reduces the storage required as well as making it very easy to augment the available output formats and make them available, 'retrospectively', to pre-existing transcripts.

### *3.1.2. Administration functionality*

Users of the system can also have 'administration' rights which provides access to all the system's configuration and management interfaces.
*User management.* Full control is provided over any user's profile including the ability to disable the account. In addition to this, it is possible to view all their uploads using the same interface used for 'user' level upload management (Fig. 3) as well as adjust their upload quota. As mentioned in Section 3, administrators can also manage a user's deleted uploads, processes and transcripts and, if requested, undelete them. Furthermore, administrators can also initiate audio recognition using an ASR system for which that user would not normally have access permissions (see below).
*Process management.* It is useful to be able to monitor the ASR processes which are currently running in response to audio uploads. Here, administrators can view the details of the associated upload, the system being used to generate the transcript and also the current status of the job (queued, running, etc). In

addition, it is possible to stop processing.
*User group management.* Each user is associated with a single user group. It is through the use of user groups that webASR determines which ASR systems are available to particular users. For example, some systems require significantly more compute resources than others and their use is restricted to certain users. Note, however, that administrators can also create exceptions to system-access permissions on a per-user basis thus adding extra flexibility to the user group approach.
*Policy management.* Policies related to upload quotas, user groups and execution permissions can be edited or deleted; new ones can also be created.
*ASR system management.* WebASR has a number of ASR systems available for use. Here, administrators can add, edit or delete such systems. For each, a name and description can be defined as well as the system type (lapel, array, ihm, etc) as well as a clearance level. The latter is used in conjunction with the user group permissions to determine who has access to each system.
*Notice management.* At certain times, maintenance or upgrade work needs to be carried out either to the server code or the underlying infrastructure. System-wide announcements provide an easy way to distribute information or downtime warnings: the announcement is displayed in a notification box once the user has logged in.
*Documentation.* Administrators also have full access to all the internal documentation covering database and server configuration through to details of the applet and API plugins.

### 3.2. Application programming interface (API)

An application programming interface (API) specifies the precise details of how a software program or service can be accessed by another software agent. In the context of the webASR service, the webASR API specifies the way in which any software program can upload audio and retrieve the associated transcripts via HTTP with no recourse to the browser-based interface described in Section 3.1.

In many respects, the usage model of the API is a restricted version of the browser-based interface and follows the same underlying HTTP protocol.
*Authentication.* In order to access the service, the user must log in using the API-specific authentication details. Upon successful authentication, a cookie is provided which is required for all subsequent communication with the webASR service.
*Supported audio types.* Since the webASR service will only accept a finite set of audio file formats, it is necessary that the client application ensure that all files are suitable for upload. The set of acceptable file formats can be downloaded in XML format; it is necessary to perform this step regularly since supported file formats may change without notice.
*Audio file upload.* For every file sent to the webSR service (both via the API as well as the applet), file metadata must also be sent. This XML file encapsulates details such as file size, client filename, the internal audio format and an MD5 hash. The file size is used to check, server-side, that the complete file has been uploaded. The service can also be configured to use the MD5 hash to save bandwidth: if an audio file has been uploaded with the same hash (and thus almost certainly the same audio file) the file is not uploaded again, rather it is linked to the existing server-side copy. Upload of the same file is common when test data for an evaluation is released and multiple, independent, users wish to recognise it. Finally, in addition to the audio file itself, the API also provides the ability (not available via the browser interface) to upload metadata related to the audio content. This is freeform XML but is commonly used to convey segmentation information to assist the ASR process. A unique upload ID is returned upon successful upload.
*Status.* The processing status associated with a previously uploaded audio file can be determined by sending the unique upload ID. The returned status can be *queued*, *running*, *failed*, *killed* or *completed*.
*Wait time.* An estimate of the amount of audio data (in seconds) to be processed on the system gives an indication of how long a client may need to wait for their transcript.
*Transcript.* An audio transcript can be downloaded in XML format. A number of the ASR systems available to webASR are multipass; this means that partial output may be available before the final, higher accuracy, transcript is available. An HTTP response header indicates whether it is complete or not.

Integration of the API-based service is facilitated by the availability of a plugin DLL for Microsoft Windows platforms. This was written in C# and thus the source code can be integrated with any .NET application. The service can also been integrated in Mac OS X applications.

## 4. Conclusions

We have presented the webASR system which provides a free to access web-based interface to the state-of-the-art AMIDA speech transcription system. The webASR system provides two modes of access: via a standard web browser and programatically via an API. The former approach provides an easy-to-use way of transcribing audio recordings without any need for technical expertise (in either speech recognition or web services). The latter allows the service to be seamlessly integrated into applications and other services without the need to use the browser-based interface. For the first time, the webASR service brings free speech recognition within the reach of the wider scientific community.

## 5. Acknowledgements

The authors would like to thank all contributors to webASR speech recognition systems, from the AMI consortium, or otherwise. In particular we would like to mention Vincent Wan and Asmaa El Hannani formerly of University of Sheffield; Lukas Burget and Martin Karafiat from University of Technology, Brno; Mike Lincoln from University of Edinburgh; John Dines and Phil Garner from Idiap; Thomas Niesler and Febe de Wet from University of Stellenbosch; Phil Woodland from Cambridge University; and many others from these and other sites.

This work was partly supported by the European IST Programme Project AMIDA (Augmented Multi-party Interaction with Distance Access) FP6-033812.

## 6. References

[1] J. Carletta et al., "The AMI meeting corpus," in *Proc. MLMI'05*, Edinburgh, 2005.

[2] T. Hain, L. Burget, J. Dines, P. N. Garner, A. El Hannani, M. j. Huijbregts, M. Karafiat, M. Lincoln, and V. Wan, "The AMIDA 2009 Meeting Transcription System," in *Interspeech'10*, 2010, pp. 358–361.

[3] T. Hain, A. el Hannani, S. Wrigley, and V. Wan, "Automatic speech recognition for scientific purposes - webASR," in *Interspeech'08*, 2008, pp. 504–507.

[4] T. Hain, L. Burget, J. Dines, M. Karafiat, D. van Leeuwen, M. Lincoln, G. Garau, and V. Wan, "The 2007 AMI(DA) system for meeting transcription," in *Proc. NIST RT07 Workshop*, 2007.