RTTS: Towards Enterprise-level Real-Time Speech Transcription and Translation Services

Juan M. Huerta¹, Cheng Wu¹, Andrej Sakrajda¹, Sasha Caskey¹, Ea-Ee Jan¹, Alexander Faisman¹, Shai Ben-David⁵, Wen Liu¹, Antonio Lee⁴, Osamuyimen Stewart¹, Michael Frissora¹, David Lubensky³

¹ IBM T J Watson Research Center
² IBM Haifa Research Lab
³ IBM China Research Lab

{huerta,chengwu,ansa,sashac,ejan,alexf}@us.ibm.com

Abstract

In this paper we describe the RTTS system for enterprise-level real-time speech recognition and translation. RTTS follows a Web Service-based approach which allows the encapsulation of ASR and MT Technology components thus hiding the configuration and tuning complexities and details from the client applications while exposing a uniform interface. In this way, RTTS is capable of easily supporting a wide variety of client applications. The clients we have implemented include a VoIP-based real time speech-to-speech translation system, a chat and Instant Messaging translation system, and a Transcription Server, among others.

Index Terms: Automatic Speech Recognition, Speech Translation Systems, Speech to Speech Translation systems, Speech and Language Web Services

1. Introduction

It is widely accepted that Speech and NLP technologies have reached a point of maturity sufficient to support their combination into complex applications [1]. A relatively unexplored question is how these technologies should be integrated into systems aimed to provide support to enterprise level applications. In this paper we describe the RTTS system (Real Time Transcription and Translation Services) which is a Web Service-oriented system that integrates these language technologies in support of real time enterprise-level speech and text-based language processing services. While today there already exist systems that jointly leverage speech recognition and machine translation technologies in portable and workstation oriented systems (e.g., Verbmobil [3], IBM Multilingual Automatic Speech to Speech Translator (MASTOR [4]), server-based Web-Service accessible systems (e.g., GALE [1]) present several relative advantageous qualities. Specifically, they can

1. Access larger computational resources and thus support large models for LVCSR recognition and statistical machine translation,
2. Support the processing of a wide range of multiple formats and communications protocols for audio, text, video, and combinations of these modalities,
3. Attain better reliability, configurability, multuser support, load balancing and CPU utilization, and
4. Present a simpler API to client applications.

While GALE [1] systems are server based systems, with RTTS our focus is on providing support to a wide range of enterprise clients in Real Time while emphasizing system scalability.

This paper is organized as follows: In Section 2 we introduce the RTTS system and describe its components, architectural organization and services it provides, in Section 3 we briefly discuss some usability considerations for real-time speech to speech applications as well as text based translation, in Section 4 we describe the client applications we have implemented around RTTS and in Section 5 we describe possible directions for future work.

2. RTTS

The RTTS system provides access to translation and speech services allowing for the construction of communication channels (email, chat and phone, for example) with integrated recognition and translation engines working on the content of the interchange. The basic translation service can be seen as a text-to-text converter accessible as a Web service, for example, using the Representational State Transfer (REST), with sufficient flexibility to accommodate chat, instant messaging, email and textual part of the speech-to-speech (S2S) exchange. The S2S exchange is performed through two possible approaches, one is IP telephony channels utilizing ASR and TTS in addition to translation via SIP/RTP, the other is to use the REST interface voice data as a MIME data attachment. From the application point of view, RTTS presents two modalities of operation: on-line real time applications (e.g., Speech 2 Speech via SIP/RTP) and off-line batch applications (e.g., wave file transcription). The RTTS architecture allows for extending the range of offered services driven by granular requests with textual or media stream data and producing the results as text or media stream. The RTTS provides merely interfaces to establish a session and data processing requests. The RTTS platform represents a scalable, redundant voice/text enterprise-level service delivery platform designed to provide high levels of availability in a cost-effective manner.

2.1. RTTS Architecture

Figure 1 shows the architectural implementation of RTTS. As we mentioned above, any external applications (clients) access their services over Internet using either the Representational State Transfer (REST) interface or through the SIP/RTP request servlet interface which is capable of handling VOIP calls. Both interfaces (REST and SIP/RTP servlet) are packaged into a web application installed in the host. The Dialog Manager (Session Manager) is responsible for coordinating allocation of resources and progress of a session. The handling of RTP streams associated with SIP calls is delegated to the Media Controller...
(MC) which is a standalone Java process. The requests submitted to the RTTS REST interface may contain an audio attachment (or reference to audio to be retrieved) and they can ask for audio output to be returned over the web connection. In this case the streaming is handled by the Dialog Manager. Streaming from the Dialog Manager and Media Controller to the ASR Servers and from the TTS servers takes place over point-to-point connections over UDP (for real-time) and TCP (for batch mode) sockets. The control data defining attributes of the request (languages, domain, etc.) is distributed over UIMA [2] infrastructure based on a robust Message Oriented Middleware layer (JMS) which ensures reliability and robustness. Thus, the ASR, translation or analytic engines that are used in RTTS must be wrapped as UIMA annotators.

The requests submitted to RTTS by client applications are passed to the Dialog Manager (DM) which, based on the content of the request, allocates necessary resources (e.g., a pair of Media Controller channels—one RTP/RTCP and one stream) and dispatches requests for specific service over UIMA (which uses in turn JMS). The UIMA request contains a URL defining the streaming endpoint (either the Media Controller or the Dialog Manager). Prior to submitting a UIMA service request, the Session Manager checks the Translation Memory (TM) and the Translation Cache handler to verify if a text translation is available. The combination of Translation Memories and Statistical Machine Translation has been previously shown to be advantageous from accuracy and response time perspectives [5]. Translation Caches, on the other hand, help avoid translating the same material repeated times helping the overall performance.

Figure 2, shows the RTTS Component Connection diagram. In terms of inter-component communication, the following methods are used:

- **Intra-thread communication**: asynchronous event mechanism built around Java new IO (NIO) using pipes for notifications and dynamic queues for messages,
- **Inter-process communication**: the control data passed between the SIP Handler and the Session Manager as well as the Session Manager (Dialog Manager (DM)) and media Controller uses TCP sockets with serialized tag/value pairs,
- **Speech and Translation Services access**: the control data passed to Services from Session Manager (DM) utilizes UIMA mechanism built on top of JMS, and
- **Streaming**: the audio streaming is implemented over IP sockets (UDP for real-time, TCP for batch mode); the audio is passed in packets with a header used to assure synchronization; the validation includes verification of session id, request number and chunk number within request.

Figure 2. RTTS Component Connection Diagram

The complexity of the session management and user interactions varies depending on the selected mode of operation from fairly trivial request-response in the case of short message translation to rather complex for two-channel speech-to-speech (S2S) session. In summary, access to RTTS services is possible through:

- SOAP/HTTP-based Web Services or REST-based interfaces. The available services as well as the specific properties of the services (e.g., language pairs, acoustic models or language models, or protocols supported) can be determined via the Discovery Service.
- SIP interface to initiate, modify or terminate a session to support speech-to-speech translation via VoIP. Speech to speech translation can also be implemented using a coordinated session consisting of a data channel submitting web service (REST) requests and one or two VoIP calls.

### 2.2. Translation and Recognition Services

This section describes the language services that are available to a client application developer. The currently implemented services of RTTS are performed by combining any of these three services:

- **Automatic Speech Recognition (ASR)**: This component performs the task of speech to text conversion. The Statistical Language Model (SLM) activated during recognition is language specific and may be domain specific too. It may be augmented by embedded grammars (dictionaries extending the accepted vocabulary). Multiple engine instances are configured to address various language-SLM domain combinations.
- **Translation Engine**: This component performs the translation of normalized text in one language to normalized text in another language. The engine is based on statistical machine translation technology. The statistical models used by the translation engine may be domain specific for improved accuracy.
• **Text to Speech (TTS):** This component performs the task of text to speech conversion for a specific language. Both the text and speech are in the same language.

All services engines are wrapped into UIMA Annotator shells and appear as UIMA components requesting tasks from the JMS queue. The engines typically consume significant resources at run-time, both in terms of memory and CPU. ASR and TTS, given enough resources, can operate in near real-time (i.e. ASR can process data from the audio stream with fixed throughput and latency close to the segment duration, TTS can produce audio faster than the speed of output stream given enough memory, CPU speed, network bandwidth etc.).

To support multiple concurrent users, RTTS is built to run in a multi-server cluster leveraging JMS. All services are allocated to user sessions only when needed and are released upon the completion of each single request. ASR and Translation services may be based on different engines for a specific application (currently, RTTS provides access to 3 language pairs: English ⇔ Chinese, English ⇔ Spanish, and English ⇔ Iraqi).

### 3. Usability Considerations

RTTS naturally raises a broad set of usability questions at several levels. At the lowest level, the most important issues are related to how to handle transcription and translation for real-time clients under imperfect accuracy and non-negligible latency. While it is better to let the client applications decide to how to handle regions of low confidence and rejection, the RTTS system provides two byproducts in support of this client feature: an overall confidence score, and a round-trip translation. The round trip translation can be presented to the user as a way to identify potentially erroneous translations allowing the user to repeat or rephrase.

ASR and MT engines can be tuned to provide high throughput (measured in xRT) yet ASR latency will unavoidably depend principally on the length of the processed segment (e.g., up to a few seconds). Shorter segments support better latency at the expense of accuracy. Thus, most of the clients prefer a “push to talk” half duplex communication which permits larger latencies. To improve translation throughput, translation caches and translation memories are used (as described in Section 2). This is important, given that a large portion of the content that RTTS translates has already been translated before. Typically only 35% of the content sent for translation is unique (e.g. has not been seen before by the system). This is probably due to the fact that most translation requests come for web page translation requests and many users are interested in similar/same web pages. The average time it takes to complete a translation request of 1 document segment is around 100 ms while the cache retrieval of the same data takes on average 3ms. We noticed that with as little as 20k entries in the cache we would be able to recall at least 50% of requests. To improve the quality of translation (as well as the speed) we implemented translation memories as well. Translation memories refer to a set of translation pairs which were previously produced by humans. RTTS checks each translation request it receives to see if it exists in the memory, and if it does it retrieves that translation, otherwise it checks the translation cache.

Other high level usability considerations can be taken into account (e.g., TTS characteristics, particular communication-specific language use (e.g., emoticons, acronyms, etc.)) based on application dependent requirements. For examples see [6].

### 4. Client Applications

We now describe the RTTS clients we have implemented.

#### 4.1. VoIP Client

We implemented a S2S translation client using VoIP and accessing RTTS through a SIP call to a pre-assigned configurable number which predefined configuration (source and target language, domain, etc.). Figure 3 shows the client and a transcript of a sample English-Chinese conversation.

The audio input is streamed over RTP and the response – TTS speech based on translated result of recognition result – is played back to the client over RTP. The client needs to coordinate the sequence of events: capture audio from speaker A, accessing RTTS going from language 1 to language 2, and then playing back the resulting audio in language 2 to user B. The process is inverted when user B talks. In addition to this pure VoIP based approach, speech to speech translation can also be implemented using a coordinated session that also includes a data channel used for submitting web service requests and one or two VoIP calls. The audio input and output are streamed over RTP and text (recognition, translation and optional back translation) are retrieved over the data channel interface. A typical English ⇔ Chinese speech to speech translation example is demonstrated as the following snapshot of output of RTTS in Figure 3.

![Figure 3. VoIP S2S client application](image)

#### 4.2. SMS and Midlet Clients

We have also prototyped a couple of cell phone applications that utilize the services of RTTS. The first application is a text translation application using the SMS (Short Message Service, the second a S2S application over the cell phone’s data channel. For the SMS, We implemented a local SMS gateway which allows incoming text SMS messages to be forwarded to an application via an HTTP request. We created a Web servlet front end to handle the SMS requests and interface to the RTTS backend. The servlet parses the SMS message and makes the appropriate calls into RTTS to request translation from a source to a target language of the provided message. When the servlet receives the translation,
it returns it in the HTTP-request response back to the gateway which in turn sends the translated message back to the user who sent the original SMS message. There is a 140-byte limitation on the size of SMS messages so long messages need to be broken up by the user or by the SMS application.

The second mobile application is a speech-to-speech (S2S) translator. We built our S2S application using a Java ME (a.k.a. J2ME) MIDlet and deployed it on a couple of cell phones (a Motorola Rzor V3xx and a Blackberry). Figure 4 shows a screenshot of the Blackberry client. These clients use the native audio recording capability of the cell phone to record the user’s speech and send it via the data channel of the cell phone to the RTTS server. This is an asynchronous approach to S2S. This application can be run in either automatic or manual mode. In manual mode, we allow the user to fix any ASR errors before sending the recognized text back to the server for translation. In automatic mode, the entire transaction can take place within a single request, that is, we send audio and we get back translated text plus TTS audio.

4.3. HTTP Client for Chat and Instant Messaging

Figure 5 shows our RTTS chat client based on the Lotus Same Time chat application (which is an Eclipse-based client). The Same Time Plug-in client communicates with RTTS through its REST interface. At startup, the client queries the server for a list of supported languages and domains (through SOA calls) and makes those resources available to its users. The Same Time client provides immediate feedback to the user by showing them what the translation looks like as they type the input (it achieves this by constantly sending translation request of partial sentences to RTTS). If the user doesn’t know the other language, it also provides the ability to show real time “back translation” which is the translation of the translated output back in to the original language. This mechanism provides a way to gauge the quality of the translation. Our client also enables the user to make manual corrections to the translated output.

4.4. Transcription Server

The Transcription Server is a Web application intended for the ASR transcription of long media files (e.g. lectures, broadcast media, movies, etc.) accessible through file upload or http. In this client, a user submits a transcription job to the Transcription Server which in turn retrieves the data to a local file system, performs the necessary audio extraction and decoding (ETL process: extract, transform, load) and sends the corresponding request to RTTS using a URL to identify the extracted audio. In this case, RTTS invokes a UIMA annotator that combines audio segmentation, ASR transcription and collates the resulting ASR transcripts back together. Because transcribing large files can take substantial amount of time, the user is notified of the status of the job through the application’s Web interface or through email upon completion. Figure 6 shows a diagram of the application.

5. Conclusions

In this paper we have described RTTS which is a system for real time transcription and translation services. RTTS makes ASR, MT and TTS available through a Web interface as well as a VoIP interfaces via SIP/RTP and provides a uniform interface in support for a diverse set of client applications. We have illustrated the simplicity entailed in implementing these clients. Currently, applications implementing T2S and S2T for transcription and translation, as well as T2T and S2S translation services have actually been piloted with real enterprise customers. Key usability issues that require future research include further improving the overall latency and throughput of the system especially in ASR and Machine Translation as well as client independent error identification, recovery and correction mechanisms.

6. References