xLiMe
FP7-ICT-2013-10
ICT-2013.4.1
Contract no.: 611346
www.xlime.eu

Deliverable D2.1.2

Final Speech to Text Prototype

Editor:	Blaž Novak, JSI
Author(s):	Blaž Novak, JSI
Deliverable Nature:	Report (R)
Dissemination Level:	Public (PU)
Contractual Delivery Date:	M24 – 31 October 2015
Actual Delivery Date:	M24 – 31 October 2015
Suggested Readers:	All project partners
Version:	1.0
Keywords:	Speech recognition; ASR; text; extraction; API; streaming

Disclaimer

Full Project Title:	xLiMe – crossLingual crossMedia knowledge extraction
Short Project Title:	xLiMe
Number and Title of Work package:	WP2 Text Extraction from Multilingual Multimedia Natural Language
Document Title:	D2.1.2 – Final Speech to Text Prototype
Editor:	Blaž Novak, JSI
Work package Leader:	Blaž Novak, JSI

Executive Summary

In this deliverable we describe the final outcome of the “speech to text” task of WP2, which provides the rest of the project pipeline with textual content based on Zattoo TV channels.

We have upgraded the prototype system that was already running after the first year and is described in the deliverable D2.1.1. Some information from that deliverable is summarized here, to make this report easier to follow.

Based on the results of last year preliminary tests, which have shown that speech recognition requires significant CPU resources and has problems with recognising a large fraction of named entities, we have decided to use any subtitles that are available from content providers, which has significantly increased the amount of quality text available to annotation services.

We also report on the evaluation of the speech recognition services used by the project, where we show that speech recognition works sufficiently well to be used as a source for knowledge extraction, but can still be much improved.

Table of Contents

Abbreviations

AAC	Advanced Audio Coding
AES	Advanced Encryption Standard
API	Application Programming Interface
ASR	Automatic Speech Recognition
CDN	Content Delivery Network
DASH	MPEG DASH - Dynamic Adaptive Streaming over HTTP
DVB	Digital Video Broadcasting
EPG	Electronic Program Guide
FTP	File Transfer Protocol
GUID	Globally unique identifier
H.264	MPEG-4 Part 10 Advanced Video Coding
HDS	HTTP Dynamic Streaming
HLS	HTTP Live Streaming
HTTP	Hypertext Transfer Protocol
JSON	Javascript Object Notation
MSF	Vecsys MediaSpeech
RDF	Resource Description Framework
SOAP	Simple Object Access Protocol
TTML	Timed Text Markup Language
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
XML	eXtensible Markup Language
ZAPI	Zattoo API

1 Introduction

The purpose of the task T2.1 is to deal with audio components of the multimodal data available to the project. All the audio data comes from TV channels that are provided by the project partner Zattoo. Zattoo is an Internet TV service provider for European viewers, and has given xLiMe access to all of its content.

In the previous year, we have developed a system that continuously monitors a selected set of TV channels, downloads the audio content, uses external speech recognition services, performs basic text annotation, and delivers the annotated text to other work packages through a Kafka message queue, in close to real time – with at most 2 minutes of latency. Because the use cases focused on news content, we picked a subset of English, German, Italian and French TV channels that mostly show daily news. Part of the software developed in T2.1 was also used in T2.2, to download the related video content.

In the deliverable D2.1.1, we have identified the need for better coverage of German language, lower system latency, increase in the amount of audio data processed, optimization of various parameters used to decide on how to do speech recognition, and overall evaluation of the system according to requirements of the dependent work packages.

This year, we made gradual improvements to the entire system according to requirements described in D2.1.1, and implemented some new functionality to adapt to new use case requirements. During the year, use case related to eConda expressed interest in having access to audio from non-news TV channels, and the Zattoo use case pivoted to recommending their live TV content to German speaking readers of news articles on affiliated websites, which demands even lower latency from the transcription system.

Based on preliminary evaluations of accuracy and speed, we decided that using only automatic speech recognition will not be sufficient. In the beginning of the year, Zattoo set up an experimental web-service that provided access to subtitles from one German TV channel. During the year, coverage was extended to a larger set of channels and languages. Recently, they started providing their TV streams using a new protocol, which includes support for transport of subtitles. We have implemented a protocol client to access those subtitles, first using the experimental web-service and now using the new DASH protocol. We implemented filtering steps to clean up the subtitles, and stream them to Kafka consumers in a manner compatible with text extracted using speech recognition services. This means zero latency and complete coverage for subtitled TV programs, and more speech recognition resources available for the remaining programs.

2 System overview

The system consists of multiple components connected together via ZeroMQ [1] message queue, as depicted on Figure 1.

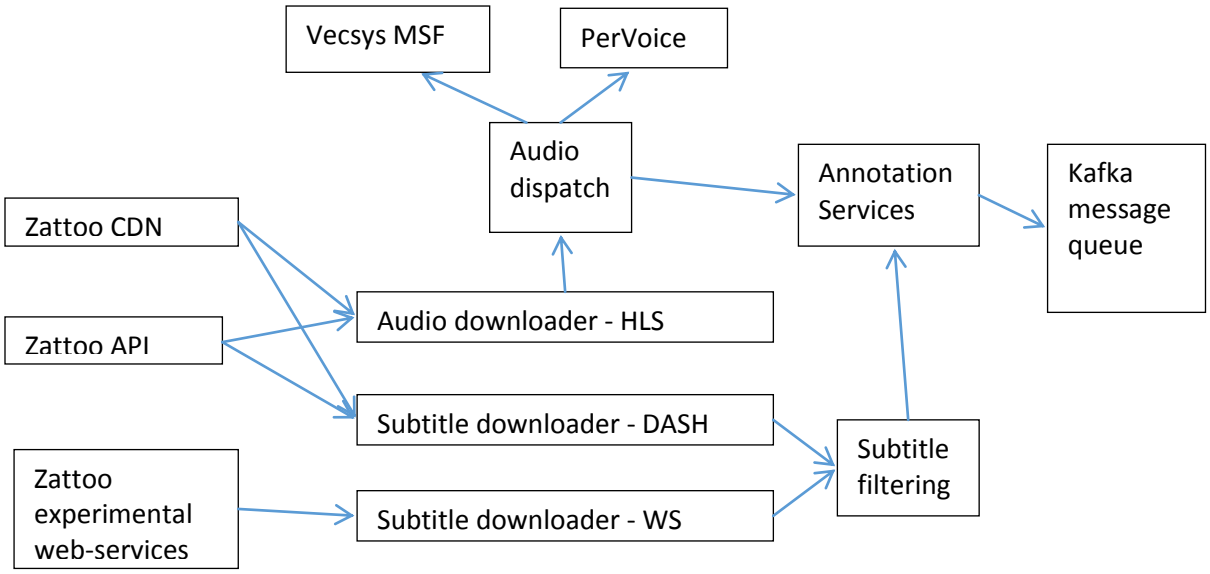

Figure 1: Overall system design

All operations that include access to Zattoo data need to go through an API endpoint on the Zattoo servers, called ZAPI. The exceptions to this are the web-service based subtitle downloader, which is now being phased out and was only an experimental service provided to xLiMe, and actual video data download, which is performed using standard HTTP GET methods issued to servers on the Zattoo content delivery network. To decrypt the video data, an encryption key is acquired through ZAPI.

The audio download subsystem has been optimized to use multiple processes to reduce latency. This forwards the raw audio to the dispatch service, which abstracts the speech recognition service APIs provided by external providers and takes care of timing synchronization.

Subtitles are downloaded by a separate service, so that we can completely omit audio data for some channels, and still have useful text.

Both subtitles and transcribed text are sent through preliminary annotation service, that extracts named entity mentions. The results are submitted to the Kafka queue in the RDF format as specified by deliverable D1.4.

The detailed description of Zattoo API, APIs to external speech recognition services, and the initial design of the entire system was provided in the deliverable D2.1.1 [2].

Table 1 shows a list of TV channels that are being downloaded and processed. There are currently 36 TV channels selected for processing, either for subtitle download, or for automatic speech recognition, compared with 5 main and 12 supplementary channels last year.

Title	Language	Audio source	Subtitle source
3Sat	German	Yes	Yes
Al Jazeera English	English	Yes	No
ARD	German	No	Yes
ARD Alpha	German	Yes	Yes
arte	German	Yes	Yes
BBC1	English	No	Yes
BBC World	English	Yes	No
Bloomberg Europe	English	Yes	No
BR	German	No	Yes
Canal 24 Horas	Spanish	Yes	Yes
CNBC	English	Yes	No
CNN International	English	Yes	No
Deutsche Welle	English	Yes	No
EuroNews	English	Yes	No
EuroNews	French	Yes	No
France24	English	Yes	No
France24	French	Yes	Yes
HR	German	Yes	Yes
ITV1	English	No	Yes
Joiz	German	Yes	No
MDR	German	Yes	Yes
N24	German	Yes	No
NDR	German	Yes	Yes
ORF1	German	No	Yes
ORF2	German	No	Yes
RAI News	Italian	Yes	No
RBB	German	No	Yes
SFR Info	German	Yes	Yes
Sky News International	English	Yes	No
SRF1	German	No	Yes
SRF2	German	No	Yes
SWR	German	Yes	Yes
tageschau24	German	Yes	No
WDR	German	Yes	Yes
ZDF	German	Yes	Yes
ZDF Info	German	Yes	No

Table 1: List of TV channels currently being downloaded

3 Subtitle streaming

3.1 Data ingestion

Subtitles for TV programs are provided by content creators. Some provide subtitles for nearly all of their programing, some provide them on a program by program basis, and some do not provide any subtitles at all.

Subtitles are sent along with the audio-video data, in the same MPEG transport stream. Zattoo converts subtitles found in the DVB stream into TTML [3] format, which stands for “Timed Text Markup Language” and is a W3C standard.

Zattoo provides multiple different protocols to access the program’s multimedia data. When we were developing the early prototype, the available options were HDS and HLS. HLS, which we chose for download of audio and video data, is implemented as a continuously updated playlist of URLs that point to 4 second long video clips. The video player client needs to download the files over HTTP, decrypt them using a key found in the playlist file, and play them in sequence.

During the last year, Zattoo included MPEG DASH (*MPEG Dynamic Adaptive Streaming over HTTP)* in the list of the supported protocols. DASH allows clients to seamlessly and dynamically switch between bitrates depending on the available bandwidth, but also includes the option of supplementary streams. Multiple audio languages can be included, as well as subtitle streams.

After issuing a *watch* ZAPI call with a parameter specifying the DASH protocol, the client is provided with an URL to an XML file called *Media Presentation Description* (MPD), which describes how audio, video and subtitle streams should be accessed. An abbreviated example of such a file is in Listing 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="urn:mpeg:dash:schema:mpd:2011"
 xsi:schemaLocation="urn:mpeg:dash:schema:mpd:2011 http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-DASH_schema_files/DASH-MPD.xsd"
 minBufferTime="PT1S"
 profiles="urn:mpeg:dash:profile:isoff-main:2011"
 type="static"
 mediaPresentationDuration="PT300S">
  <Period duration="PT300S" start="PT0S">
   <AdaptationSet id="0" mimeType="video/mp4" segmentAlignment="true" startWithSAP="1">
    <Role schemeIdUri="urn:mpeg:dash:role:2011" value="main"/>
    <SegmentTemplate presentationTimeOffset="0" media="video$Number$-$Bandwidth$-0.m4s?z32=NVQ" initialization="video-$Bandwidth$-0.mp4?z32=NVQ" startNumber="1" timescale="1000" duration="4000"/>
    <Representation id="0" codecs="avc1.4d401f" width="1280" height="720" frameRate="25" bandwidth="2800000"/>
    <Representation id="2" codecs="avc1.4d4015" width="512" height="288" frameRate="25" bandwidth="700000"/>
    <Representation id="3" codecs="avc1.4d4015" width="512" height="288" frameRate="25" bandwidth="500000"/>
    <Representation id="5" codecs="avc1.42c00b" width="400" height="224" frameRate="25/4" bandwidth="85000"/>
   </AdaptationSet>
   <AdaptationSet id="1" lang="eng" mimeType="audio/mp4" codecs="mp4a.40.2" segmentAlignment="true" startWithSAP="1">
    <AudioChannelConfiguration schemeIdUri="urn:mpeg:mpegB:cicp:ChannelConfiguration" value="2"/>
    <SegmentTemplate presentationTimeOffset="0" media="audio$Number$-$Bandwidth$-0.m4s?z32=NZQ" initialization="audio-$Bandwidth$-0.mp4?z32=NZQ" startNumber="1" timescale="1000" duration="4000"/>
    <Representation id="6" bandwidth="128000"/>
   </AdaptationSet>
   <AdaptationSet id="2" lang="fre" mimeType="audio/mp4" codecs="mp4a.40.2" segmentAlignment="true" startWithSAP="1">
    <AudioChannelConfiguration schemeIdUri="urn:mpeg:mpegB:cicp:ChannelConfiguration" value="2"/>
    <SegmentTemplate presentationTimeOffset="0" media="audio$Number$-$Bandwidth$-1.m4s?z32=NZQ" initialization="audio-$Bandwidth$-1.mp4?z32=NZQ" startNumber="1" timescale="1000" duration="4000"/>
    <Representation id="7" bandwidth="128000"/>
   </AdaptationSet>
   <AdaptationSet id="3" lang="eng" mimeType="application/mp4" codecs="stpp">
    <Role schemeIdUri="urn:mpeg:dash:role:2011" value="main"/>
    <SegmentTemplate presentationTimeOffset="0" media="subs$Number$-3000000-0.m4s?z32=NZQ" initialization="subs-3000000-0.mp4?z32=NZQ" startNumber="1" timescale="1000" duration="4000"/>
    <Representation id="8" bandwidth="100"/>
   </AdaptationSet>
```

```
 </Period>
</MPD>
```

Listing 1. – Example of DASH Media Presentation Description file, with keys removed

The MPD file specifies a template how the URLs for all stream types should be generated by the client. Like with HLS, media files are split into small, self-contained files, which can be downloaded over HTTP, and the description file specifies the duration and offset of those files.

Subtitles are provided in the XML TTML format. Listing 2 shows an example of such a file. Each tag denotes a single line of subtitles to be displayed.

```
<?xml version="1.0" encoding="utf-8"?>
<tt xml:lang="de" xmlns="http://www.w3.org/ns/ttml"
 xmlns:tts="http://www.w3.org/ns/ttml#styling">
 <head>
  <styling>
   <style xml:id="s1" tts:color="white"/>
  </styling>
  <layout>
   <region xml:id="r1" tts:extent="80% 20%" tts:origin="10% 70%" tts:textAlign="center" tts:wrapOption="noWrap"/>
  </layout>
 </head>
 <body>
  <div region="r1">
   <p xml:id="sub1446053004000" begin="401681:23:24.00" end="401681:23:25.62">
    <span style="s1">Das ist also eine Attrappe.</span>
   </p>
   <p xml:id="sub1446053004001" begin="401681:23:25.62" end="401681:23:28.00">
    <span style="s1">Aber macht es das denn besser?</span>
   </p>
  </div>
 </body>
</tt>
```

Listing 2: Subtitle TTML XML file

3.2 Filtering

The subtitle file is meant to be rendered on the screen for human viewers. This means that we often cannot simply concatenate all the text found in the subtitle file together, but need to discard part of it that was already shown.

The simplest type of filtering is applied in cases where entire lines scroll up on the screen – in such a case, the maximally overlapping set of lines from two consecutive subtitle files is found and discarded before concatenating the newly received lines with the text waiting to be sent to consumers.

In some cases of live TV, subtitles are transcribed by the studio and sent immediately when a new word is spoken. When watching TV, this appears as words being added to the last subtitle line. In such a case, a new subtitle file is sent after each word has been transcribed, and it includes not only the last word, but everything that is currently being displayed. Mistakes made during the transcription are also corrected in real-time, which means that occasionally lines are sent that contain less words than in the previous file, or even just with some words replaced. Paragraph IDs, as specified in the XML file, do not stay the same during the time one line is being corrected.

We have implemented a simple filtering solution that covers most such cases. First, all lines *above* the last one are filtered by comparing entire lines to the history buffer, not taking into account the last line of the history buffer, and marking them if they repeat. Then, the entire block of currently displayed text is compared to the history buffer, and if more than 2/3 of the words in the beginning of the last line in the block match, we discard the corresponding match in the history buffer and replace it with the new line.

Otherwise, the new line is assumed to be similar to the last one, but not the same, and is just appended to the history buffer.

After we have collected 40 seconds of subtitles – based on the timestamps found in the subtitle paragraph attributes – we collect them together and submit them for annotation. We decided on 40 second blocks to make it easy for the rest of the pipeline to ingest subtitle data, as it is already processing transcribed speech in identical block sizes.

We annotate the subtitle text using the same named entity annotation services as used by the ASR module to maintain compatibility, as described in deliverable D2.1.1. These annotation services were developed during the EU FP7 project XLike [4].

3.3 Subtitle use

Subtitles are currently used by the Zattoo use case in WP7, and by JSI EventRegistry [5] system.

In the Zattoo use case, subtitles are collected from the Kafka message queue, and used identically as the ASR data. They are compared to a subset of recent news articles, that are provided by the JSI NewsFeed [6] system through the same Kafka system, and come from a set of whitelisted sites that are affiliated with Zattoo. Video recommendations are then provided to the readers. Deliverable D7.2.2 describes this use case.

JSI EventRegistry is a system that tries to extract and track events based on current published news. It is being used and extended for some tasks in the xLiMe project. We have recently added the capability to treat chunks of subtitle text as if they represent news articles, allowing video news content to be linked to the same events as textual news. For efficiency reasons, we have implemented a bypass path for subtitle data from the subtitle download system to the EventRegistry system, since they both run on the same physical network. We will redesign this when other work-packages begin providing additional annotations for subtitles.

4 Speech recognition improvements

4.1 Overview

Automated speech recognition is the primary purpose of the task T2.1. It has been operational since last year, but we have implemented several improvements.

Audio content is downloaded from the Zattoo CDN network in small segments of 4 seconds. To improve the recognition performance, we collect multiple segments together, and submit them for speech recognition. We have initially picked 40 seconds as a trade-off between transcription accuracy due to large context, and latency, which increases with increasing buffer size.

This year, we have performed some evaluation of the effect of the chunk size on the transcription performance, and while it seems that there is little appreciable difference in performance between 40 and 20 second chunks, as described in the following sections, we kept the chunk size at 40 seconds, because the overhead of calling external speech recognition services reduces the benefit that shorter chunks have on latency, while changing the chunk size would at the same time require change of message consumers in other work packages.

We used two speech recognition services for D2.1.1 – Vecsys MediaSpeech and PerVoice in an experimental fashion. We wanted to include more speech recognition systems in the evaluation this year, such as the open-source CMU Sphynx [7], and Microsoft Speech API [8], but we found out that all of the freely available systems require training data in the form of pronunciation dictionaries and language models, which we would have to create by ourselves. Evaluation is thus limited to the same two services as have been in use since the beginning. The evaluation shows the necessity of maintaining an up to date dictionary and language model.

4.2 Performance improvements

4.2.1 Speed

In D2.1.1, we have determined that the main limiting factor of the speech recognition system was its throughput. At the time, we were only using the official Vecsys cloud instance of MediaSpeech service. We were also evaluating the PerVoice system on English audio, but the results were not yet forwarded to the Kafka message queue.

In cooperation with Vecsys, we set up three virtual machines on KIT servers, and got the MediaSpeech software running on them. There is a difference in behaviour of the cloud instance and private instances of the speech recognition software, however. The cloud version of the service allows us to upload audio data via an FTP channel, as described in D2.1.1, whereas the private instances did not allow this; they are only able to download audio data from a webserver.

To feed the audio data to KIT MediaSpeech servers, we installed a database for audio clips at JSI, where all of the audio data from tracked channels gets stored for a certain time period – a week by default. A web-service then provides this data, appropriately concatenated, to our MediaSpeech instances.

After setting up the system, we tested it, and found out that the transcription takes almost five times as long as when running in the cloud. After consulting with Vecsys, it was determined that the likely cause is the difference between CPUs in their cloud servers and the KIT provided virtual machines.

Transcription of 40 seconds of audio currently requires on average 125 seconds when running on the cloud instance, and approximately 550 seconds when running on KIT servers. Since use cases require near real-time performance, we have disabled the use of KIT instances, reverting back to cloud service.

MediaSpeech software did not support German language when we started using it, but Vecsys added the capability during this year.

As an alternative to the MediaSpeech service, we started using the PerVoice system. PerVoice is a distributed service, with worker servers deployed around various institutions and companies. We performed all of the evaluation which is described in the next section on their cloud instances, but since it is already designed as a distributed system. It will allow us to run worker machines as a part of the project, and use the same interface that we are using now, again as described in D2.1.1. We are currently in the process of deploying worker services on 10 virtual machines provided by the KIT Supercomputing Centre.

Unlike the Vecsys MediaSpeech software, which processes audio only in blocks of finite size, PerVoice performs transcription on a stream of raw audio data.

With the current configuration of the software, which determines the trade-off between accuracy and speed, PerVoice service runs at approximately 62% of real-time speed -- i.e., it takes 160 seconds to process 100 seconds of audio data. Because it takes 10-20 seconds to start a PerVoice client process, we have modified our audio distribution system in such a way that it maintains a pool of connections to the PerVoice service cloud. We split TV audio into similar 40 second chunks as we do for MSF, but send them to any available worker, regardless of the TV channel the audio chunk came from, and identity of the worker. If possible, we do reuse the same worker for the same TV channel. Because transcription workers only accept raw audio data, there is no way of notifying them that an unrelated audio segment started. To avoid introducing more errors in the transcription near the audio chunk boundaries, we insert 5 seconds of silence between unrelated chunks.

Results from the PerVoice client are timestamped, which allows us to determine which transcribed words belong to which input chunks. If successive audio chunks submitted to a single worker do not belong to the same channel, we discard 7 seconds of transcription output, to remove errors due to words that got cut up in the chunking process.

4.2.2 Accuracy

We modified the audio chunk generation so that it can produce chunks that overlap already submitted chunks, by a configurable amount of time. The overlap is recorded in the chunk metadata. When the transcription is complete, sequential overlapping chunks are combined together by searching for the longest common phrase contained in the overlapping segment, discarding the rest of the transcription in the overlapping segment, and only using a single copy of the common substring. If no common substring is found, overlapping segments are cut in the middle before merging them, to maintain the same length of time.

The purpose of this was to reduce transcription errors on audio chunk boundaries when using Vecsys MSF, however it is currently not in use, since we do not have access to sufficient CPU power for even non-overlapping transcription.

4.3 Evaluation

Based on the requirements provided by use cases, we have decided to focus on the accuracy and completeness of extracted named entities from the speech.

Since the purpose of this task was not to improve on existing annotation services, we decided not to use human annotators as the reference, but to compare annotations of extracted audio to annotations on available subtitles. We assume that subtitles are of sufficient quality to be used as the golden standard for this comparison. Other work packages are primarily interested in text annotations – i.e. extracted named entities – so we used this as the target metric. We used the XLike developed annotation service (“Wikifier”) that is also used in the course of real-time audio transcription as the named entity recognizer.

Additionally, we were interested in the already mentioned latency and throughput of the available speech recognition services, and the effect of chunking on the quality of transcription and annotation.

4.3.1 Experimental setup

We have collected a parallel corpus of subtitle and audio data in German and English and used it to compare accuracy of various speech recognition setups. The content is marked as "informational" in the electronic program guide, which means mostly news broadcasts. We are looking into releasing the extended version of the dataset publically. Table 2 gives a summary of the evaluation dataset.

Language	Number of clips	Total dataset length	Total subtitle lines	Total subtitle words
German	51	86400s	33830	139763
English	43	86400s	45597	227868

Table 2: Evaluation dataset statistics.

For all the experiments, the setup is the same: we use a speech recognition service to extract text from audio data, then we annotate this text using the selected wikifier. We compare the annotations produced by the speech recognition service with the annotations produced by running the wikifier on the subtitle data.

Both speech recognition services output timestamps for words, from which we transfer timestamps to annotations. Due to the inexact nature of the transcription process, we allow for some temporal mismatch – up to 3 seconds – and still count annotations as matching. In other words, for each annotation, we check to see if there is a matching annotation in the other dataset within up to 3 seconds of temporal distance.

We measure precision and recall, and report the micro average across all clips. Precision is calculated as the ratio of correct annotations with respect to all annotations produced on the text given by the speech recognition method, and recall is calculated as the ratio of correct annotations with respect to the number of all true annotations.

4.3.2 Results

Table 3 lists the experiments performed, with the precision and recall scores achieved, with best performing method in boldface.

	English		German	
	Precision	Recall	Precision	Recall
MSF complete	**0.39**	**0.51**	**0.30**	**0.35**
MSF 40s	0.39	0.48	0.27	0.33
MSF 20s	0.38	0.43	0.27	0.28
MSF 40s+10s	0.39	0.50	0.29	0.33
PerVoice complete	-	-	0.24	0.16
PerVoice 40s+5s	-	-	0.24	0.14

Table 3: Precision and accuracy of various speech recognition engines

Experiments listed as "MSF complete" and "PerVoice complete" mean we sent the entire audio clip (on average approximately half an hour of news segment) through the ASR service in one call. "MSF 40s" and "MSF 20s" means audio was chunked into short clips of 40 and 20 seconds respectively, and sent segment by segment, with results concatenated together. "MSF40s+10s" denotes 40 second chunks with 10 second

overlap, and “PerVoice 40s+5s” means 40 second audio chunks with 5 second of silence inserted between chunks. We used a single worker to process the entire clip.

We did not perform PerVoice experiments on English language due to issues with calling the appropriate service.

4.3.3 Discussion

From the results, we can see that the best accuracy can be had by using Vecsys MediaSpeech software with sufficiently long intervals. We suspect the lower performance from the PerVoice service is due to a different language model: MediaSpeech is, among other things, developed to process news clips, while the PerVoice instance that we are using is trained to process lecture audio. A casual look at the extracted named entities supports this hypothesis – emerging named entities are not detected at all.

Since the PerVoice system is developed at KIT, we will try to provide the developers with training data that is more suitable to the content domain that our use cases are interested in, just like we provided Vecsys with a large dump of German news articles from the JSI Newsfeed system in the previous year.

At least for processing news content, which provides large amounts of redundancy, low precision and recall can be compensated for to a certain extent by using more data, but we need to further improve transcription accuracy for other use cases.

5 Conclusion and future work

In the task T2.1 we have successfully developed a complete system that acts as an intermediary between audio data sources, speech recognition engines, and consumers within the project. Our evaluation shows that, while not perfect, audio data can be used for further knowledge extraction.

The biggest open issue is the throughput achieved, but we hope to mitigate this by deploying more transcription service instances on our servers.

In the coming year, we will continue to improve the system, focusing on the PerVoice accuracy.

References

[1] http://zeromq.org

[2] xLiMe Deliverable D2.1.1, Blaz Novak, 31. 10. 2014

[3] http://www.w3.org/TR/ttml1/

[4] http://xlike.org/

[5] http://eventregistry.org/

[6] http://newsfeed.ijs.si/

[7] http://cmusphinx.sourceforge.net

[8] https://msdn.microsoft.com/en-us/library/ms723627%28v=vs.85%29.aspx

