SUMMA: Integrating Multiple NLP Technologies into an Open-source Platform for Multilingual Media Monitoring

Ulrich Germann
University of Edinburgh
ugermann@ed.ac.uk

Renārs Liepiņš
LETA
renars.liepins@leta.lv

Didzis Gosko
LETA
didzis.gosko@leta.lv

Guntis Barzdins
University of Latvia
guntis.barzdins@lu.lv

Abstract

The open-source SUMMA Platform is a highly scalable distributed architecture for monitoring a large number of media broadcasts in parallel, with a lag behind actual broadcast time of at most a few minutes.

It assembles numerous state-of-the-art NLP technologies into a fully automated media ingestion pipeline that can record live broadcasts, detect and transcribe spoken content, translate from several languages (original text or transcribed speech) into English,[1] recognize Named Entities, detect topics, cluster and summarize documents across language barriers, and extract and store factual claims in these news items.

This paper describes the intended use cases and discusses the system design decisions that allowed us to integrate state-of-the-art NLP modules into an effective workflow with comparatively little effort.

1 Introduction

SUMMA ("Scalable Understanding of Multilingual Media") is an EU-funded collaborative effort to combine state-of-the-art NLP components into a functioning, scalable media content processing pipeline to support large news organizations in their daily work. The project consortium comprises seven acadamic / research partners — the University of Edinburgh, the Lativan Information Agency (LETA), Idiap Research Institute, Priberam Labs, Qatar Computing Research Institute, University College London, and Sheffield University —, and BBC Monitoring and Deutsche Welle as use case partners.

In this paper, we first describe the use cases that the platform addresses, and then the design decisions that allowed us to integrate existing state-of-the art NLP technologies into a highly scalable, coherent platform with comparatively little integration effort.

2 Use Cases

Three use cases drive the development of the SUMMA Platform.

2.1 External Media Monitoring

BBC Monitoring is a business unit within the British Broadcasting Corporation (BBC). In continuous operation since 1939, it provides media monitoring, analysis, and translations of foreign news content to the BBC's news rooms, the British Government, and other subscribers to its services. Each of its ca. 300 monitoring journalists usually keeps track up to 4 live sources in parallel (typically TV channels received via satellite), plus a number of other sources of information such as social media feeds. Assuming work distributed around the clock in three shifts,[2] BBC Monitoring thus currently has, on average, the capacity to actively monitor about 400 live broadcasts at any given time — just over a quarter of the ca. 1,500 TV stations that it has access to, not to mention other sources such as radio broadcasts and streams on the internet. NLP technologies such as automatic speech recognition (ASR), machine translation (MT), and named entity (NE) tagging can alleviate the human media monitors from mundane monitoring tasks and let them focus on media digestion and analysis.

2.2 Internal Monitoring

Deutsche Welle is an international broadcaster operating world-wide in 30 different languages. Regional news rooms produce and broadcast content independently; journalistic and programming decisions are **not** made by a central authority within Deutsche Welle. Therefore, it is difficult for the overall organisation to maintain an accurate and up-to-date overview of what is being broadcast, and what stories have been covered.

[1] The choice of English as the lingua franca within the Platform is due to the working language of our use case partners; the highly modular design of the Platform allows the easy integration of custom translation engines, if required.

[2] The actual distribution of staff allocation over the course of the day may differ.

Proceedings of Workshop for NLP Open Source Software, pages 47–51
Melbourne, Australia, July 20, 2018.

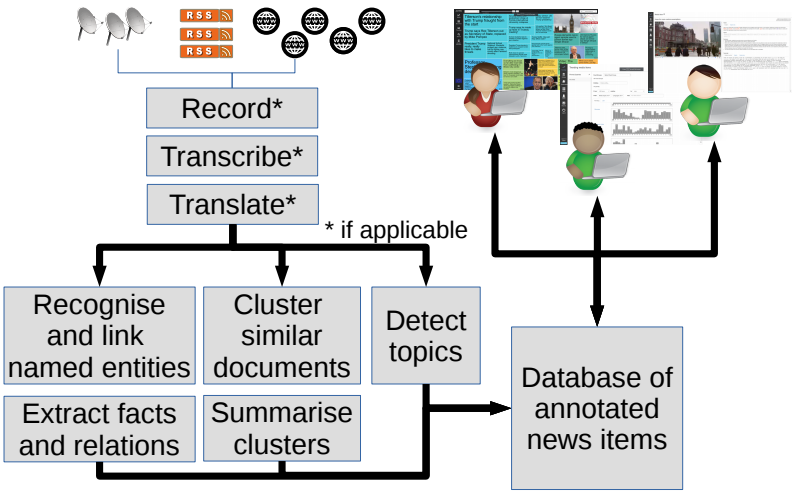

Figure 1: The SUMMA Platform Workflow

The Platform's cross-lingual story clustering and summarization module with the corresponding online visualization tool addresses this need. It provides an aggregate view of recent captured broadcasts, with easy access to individual broadcast segments in each cluster.

2.3 Data Journalism

The third use case is the use of the Platform's database of news coverage for investigations involving large amounts of news reports, for example, exploring how certain persons or issues are portrayed in the media over time.

3 System Overview

Figure 1 shows the overall system architecture. The SUMMA Platform consists of three major parts: a data ingestion pipeline built mostly upon existing state-of-the-art NLP technology; a web-based user front-end specifically developed with the intended use cases in mind; and a database at the center that is continuously updated by the data ingestion pipeline and accessed by end users through the web-based GUI, or through a REST API by downstream applications.

The user interfaces and the database structure were custom developments for the SUMMA Platform. The grapical user interfaces are based on input from potential users and wireframe designs provided by the use case partners. They are implemented in the Aurelia JavaScript Client Framework[3]. For NLP processing, we mostly build on and integrate existing NLP technology. We describe key components below.

3.1 Languages Covered

The goal of the project to offer NLP capabilities for English, German, Arabic, Russian, Spanish, Latvian, Farsi, Portuguese, and Ukrainian. We currently cover the former 6; the latter 3 are work in progress.

Due to the large number of individual components (number of languages covered times NLP technologies), it is not possible to go into details about training data used and individual component performance here. Such details are covered in the SUMMA project deliverables D3.1 (Garner et al., 2017), D4.1 (Obamuyide et al., 2017), and D5.1 (Mendes et al., 2017), which are available from the project's web site.[4] We include applicable deliverable numbers in parentheses below, so that the inclined reader can follow up on details.

3.2 Live Stream Recorder and Chunker

The recorder and chunker monitors one or more live streams via their respective URLs. Broadcast signals received via satellite are converted into transport streams suitable for streaming via the internet and provided via a web interface. This happens outside of the platform since it requires special hardware. The access point is a live stream provided as an `.m3u8` playlist via a web interface that is polled regularly by the respective data feed module.

All data received by the Recorder-and-Chunker is recorded to disk und chunked into 5-minutes segments for further processing. Within the Platform infrastructure, the Recorder-and-Chunker also serves as the internal video server for recorded

[3] aurelia.io

[4] www.summa-project.eu/deliverables

transitory material, i.e., material not obtained from persistent sources.

Once downloaded and chunked, a document stub with the internal video URL is entered into the data base, which then notifies the task scheduler about the new arrival, which in turn schedules the item for downstream processing.

Video and audio files that are not transitory but provided by the original sources in more persistent forms (i.e., served from a permanent location), are currently not recorded[5] but retrieved from the original source when needed.

3.3 Other Data Feeds

Text-based data is retrieved by data feed modules that poll the providing source at regular intervals for new data. The data is downloaded and entered into the database, which then again notifies the task scheduler, which in turn schedules the new arrivals for downstream processing.

In addition to a generic RSS feed monitor, we use custom data monitors that are tailored to specific data sources, e.g. the specific APIs that broadcaster-specific news apps use for updates. The main task of these specialized modules is to map between data fields of the source API's specific response (typically in JSON[6] format), and the data fields used within the Platform.

3.4 Automatic Speech Recognition (D3.1)

The ASR modules within the Platform are built on top of CloudASR (Klejch et al., 2015); the underlying speech recognition models are trained with the Kaldi toolkit (Povey et al., 2011). Punctuation is added using a neural MT engine that was trained to translate from un-punctuated text to punctuation. The training data for the punctuation module is created by stripping punctuation from an existing corpus of news texts. The MT engine used for punctuation insertion uses the same software components as the MT engines used for language translation.

3.5 Machine Translation (D3.1)

The machine translation engines for language translation currently use the Marian[7] decoder (Junczys-Dowmunt et al., 2016) for translation with neural MT models trained with the Nematus toolkit (Sennrich et al., 2017). We have recently switched to the Marian toolkit for training.

3.6 Topic Classification (D3.1)

The topic classifier uses a hierarchical attention model for document classification (Yang et al., 2016) trained on nearly 600K manually annotated documents in 8 languages.

3.7 Storyline Clustering (D3.1) and Cluster Summarization (D5.1)

Incoming stories are clustered into storylines with Aggarwal and Yu's (2006) online clustering algorithm. The resulting storylines are summarized with the extractive summarization algorithm by Almeida and Martins (2013).

3.8 Named Entity Recognition and Linking (D4.1)

For Named Entity Recognition, we use TurboEntityRecognizer, a component within TurboParser[8] (Martins et al., 2009). Recognized entities and relations between them (or propositions about them) about them are linked to a knowledge base of facts using techniques developed by Paikens et al. (2016).

3.9 Databases

The Platform currently relies on two databases. The central database in the NLP processing pipeline is an instance of RethinkDB[9], a document-oriented database that allows clients to subscribe to a continuous stream of notifications about changes in the database. This allows clients (e.g. the task scheduler) to be notified about the lastest incoming items as they are added to the database, without the need to poll the database periodically. Each document consists of several fields, such as the URL of the original news item, a transcript for audio sources, or the original text, a translation into English if applicable, entities such as persons, organisations or locations mentioned in the news items, etc.

For interaction with web-based user interfaces, we are using a PostgreSQL[10] database, which is periodically updated with the latest arrivals from the data ingestion pipeline. This second database was not part of the original design; it was added out of performance concerns, as we noticed at some point that RethinkDB's responsiveness tended to deteriorate over time as the number of items in the database grew. Ultimately, this turned out to be an error in the set-up of our RethinkDB instance: certain crucial fields weren't indexed, so that RethinkDB resorted to a linear search for certain operations. The current split between two databases is not ideal; however, it is operational and eliminating it is not a high priority on the current development agenda.

[5] On a marginal note, recording a single live stream produces, depending on video resolution, up to 25 GiB of data per day.

[6] https://www.json.org

[7] www.github.com/marian-nmt

[8] https://github.com/andre-martins/TurboParser

[9] www.rethinkdb.com

[10] www.postgresql.org

4 Platform Design and Implementation

As is obvious from the description above, the Platform utilizes numerous existing technologies. In designing and implementing the Platform, we had two main objectives:

1. Minimize the effort necessary to integrate the various existing technologies.

2. Keep individual NLP components as independent as possible, so that they can be re-used for other purposes as well.

In order to meet these objectives, we designed the processing pipeline as a federation of microservices that communicate with each other via REST APIs and/or a message queue.

For rapid prototyping, we defined REST APIs for each NLP module within the OpenAPI Specification Framework[11]. The suite of Swagger tools[12] associated with OpenAPI allowed us to specify REST APIs quickly and generate boilerplate code for the back-end server of each microservice. This reduced integration efforts to implementing a few back-end functions for each RESTful server — in whatever programming language the contributing partner felt most comfortable with.

In the Platform prototype, we used separate processes that act as dedicated intermediaries between the message queue[13] and each individual NLP component. As the Platform matures, we gradually moving to implementing RabbitMQ interfaces directly with the NLP processors, to eliminate the intermediaries and reduce the overall complexity of the system.

One of the great advantages of using a message queue is that it makes scaling and load balancing easy. If we need more throughput, we add more workers (possibly on different machines) that talk to the same message queues. Each type of task has two queues: one for open requests, the other one for completed requests. Each worker loops over waiting for a message to appear in the queue, popping it of it, acknowledging it upon successful completion (so that it can be marked as done), and pushing the response onto the respective queue. Workers need not be aware of the overall architecture of the system; only the overall task scheduler has to be aware of the actual work flow.

Maintaining the RESTful APIs is nevertheless worthwhile: it allows individual components to be deployed easily as a service outside of the Platform's context and workflow.

The overall NLP processing workflow is designed as an icremental annotation process: each service augments incoming media items with additional information: automatic speech recognition (ASR) services add transcriptions, machine translation (MT) engines add translations, and so forth.

Each component of the Platform runs independently in a Docker[14] application container. Similar to conventional virtual machines (VMs), Docker containers isolate applications from the host system by running them in a separate environment ("container"), so that each application can use its own set of libraries, ports, etc. However, unlike conventional VMs, which emulate a complete machine including device and memory management, the Docker engine allows containers to share the host's kernel and resources, greatly reducing the virtualisation / containerization overhead. For small-scale single-host deployment (up to ten live streams on a 32-core server), we use Docker Compose;[15] for multi-host scaling, Docker Swarm[16].

Another great advantage of the Docker platform is that many third-party components that we rely on (message queue, data bases) are available as pre-compiled Docker containers, so that their deployment within the Platform is trivial. No dependencies to manage, no compilation from scratch required, no conflicts with the OS on the host machine.

Our approach to the design of the Platform has several advantages over tigher integration within a single development framework in which contributing partners would be required to provide software libraries for one or more specific programming languages.

First, in line with our first design objective, it minimizes the integration overhead.

Second, it is agnostic to implementational choices and software dependencies of individual components. Each contributing partner can continue to work within their preferred development environment.[17]

Third, it provides for easy scalability of the system, as the Platform can be easily distributed over multiple hosts. With the message queue approach, multiple workers providing identical services can share the processing load.

Fourth, modules can be updated without having to re-build the entire system. Even live continuous upgrades and server migration can be accomplished easily by starting up a new instance of a specific service and then shutting down the obso-

[11] www.openapis.org; formerly Swagger
[12] www.swagger.io
[13] RabbitMQ (https://www.rabbitmq.com/) works well for us.
[14] www.docker.com
[15] https://docs.docker.com/compose/
[16] https://docs.docker.com/engine/swarm
[17] In the case of Windows-based software, however, licensing issues have to be considered for deployment in container environments such as Docker.

lete one.

Fifth, meeting our second design objective, the strong modularization of the Platform allows for easy re-use of components. The back-end server for each component can easily be integrated into other applications (albeit potentially requiring augmentations to the API).

5 Conclusion

We have reported on our experiences in implementing a high-performance, highly scalable natural language processing pipeline from existing implementations of state-of-the art as an assembly of containerized micro-services. We found that this approach greatly facilitated technology integration and collaboration, as it eliminated many points of potential friction, such as having to agree on a joint software development framework, adapting libraries, and dealing with software dependencies. Using the Docker platform, we are able to deploy and scale the Platform quickly.

Availability

The Platform infrastructure and most of its components are scheduled to be released as open-source software by the end of August 2018 and will be available through the project web site at

www.summa-project.eu.

Acknowledgements

This work was conducted within the scope of the Research and Innovation Action *SUMMA*, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688139.

References

Aggarwal, Charu C and Philip S Yu. 2006. "A framework for clustering massive text and categorical data streams." *SIAM Int'l. Conf. on Data Mining*, 479–483.

Almeida, Miguel B and Andre FT Martins. 2013. "Fast and robust compressive summarization with dual decomposition and multi-task learning." *ACL*, 196–206.

Garner, Philip N., Alexandra Birch, Andrei Popescu-Belis, Peter Bell, Herve Bourlard, Steve Renals, Sebastião Miranda, and Ulrich Germann. 2017. *SUMMA Deliverable D3.1: Initial Progress Report on Shallow Stream Processing.* Tech. rep., The SUMMA Consortium.

Junczys-Dowmunt, Marcin, Tomasz Dwojak, and Hieu Hoang. 2016. "Is neural machine translation ready for deployment? A case study on 30 translation directions." *International Workshop on Spoken Language Translation.* Seattle, WA, USA.

Klejch, Ondřej, Ondřej Plátek, Lukáš Žilka, and Filip Jurčíček. 2015. "CloudASR: platform and service." *Int'l. Conf. on Text, Speech, and Dialogue*, 334–341.

Martins, André FT, Noah A Smith, and Eric P Xing. 2009. "Concise integer linear programming formulations for dependency parsing." *ACL*, 342–350.

Mendes, Afonso, Pedro Balage, Mariana Almeida, Sebastião Miranda, Nikos Papasarantopoulos, Shashi Narayan, and Shay Cohen. 2017. *SUMMA Deliverable 5.1: Initial Progress Report on Natural Language Understanding.* Tech. rep., The SUMMA Consortium.

Obamuyide, Abiola, Andreas Vlachos, Jeff Mitchell, David Nogueira, Sebastian Riedel, Filipe Aleixo, Samuel Broscheit, Andre Martins, Mariana Almeida, Sebastião Miranda, Afonso Mendes, and Andrei Popescu-Belis. 2017. *SUMMA Deliverable D4.1: Initial Progress Report on Automatic Knowledge Base Creation.* Tech. rep., The SUMMA Consortium.

Paikens, Peteris, Guntis Barzdins, Afonso Mendes, Daniel Ferreira, Samuel Broscheit, Mariana S. C. Almeida, Sebastião Miranda, David Nogueira, Pedro Balage, and André F. T. Martins. 2016. "SUMMA at TAC knowledge base population task 2016." *TAC.* Gaithersburg, Maryland, USA.

Povey, Daniel, Arnab Ghoshal, Gilles Boulianne, Lukáš Burget, Ondřej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlíček, Yanmin Qian, Petr Schwarz, Jan Silovský, Georg Stemmer, and Karel Veselý. 2011. "The Kaldi speech recognition toolkit." *ASRU.*

Sennrich, Rico, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Läubli, Antonio Valerio Miceli Barone, Jozef Mokry, and Maria Nadejde. 2017. "Nematus: a toolkit for neural machine translation." *EACL Demonstration Session.* Valencia, Spain.

Yang, Zichao, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J. Smola, and Eduard H. Hovy. 2016. "Hierarchical attention networks for document classification." *NAACL.* San Diego, CA, USA.