NLP Web Services for Resource-Scarce Languages

M.J. Puttkammer*, E.R. Eiselen[+], J. Hocking* and F.J. Koen*
*Centre for Text Technology; [+]South African Centre for Digital Language Resources
North-West University, Potchefstroom Campus, South Africa
{Martin.Puttkammer; Justin.Hocking; Frederik.Koen;
Roald.Eiselen}@nwu.ac.za

Abstract

In this paper, we present a project where existing text-based core technologies were ported to Java-based web services from various architectures. These technologies were developed over a period of eight years through various government funded projects for 10 resource-scarce languages spoken in South Africa. We describe the API and a simple web front-end capable of completing various predefined tasks.

1 Introduction

With the establishment of large-scale e-infrastructures, there has been an international move towards making software available as a service. Web services are a way of exposing the functionality of an information system and making it available through standard web technologies (Alonso *et al.*, 2004). A natural language processing (NLP) web service refers to one or more technologies that focus on natural (human) speech or text and that are exposed programmatically to allow anyone with internet access, on multiple platforms, to gain access to the output of the technology. By hosting NLP web services, the development of end-user-facing applications could be facilitated in the sense that software developers and researchers get access to the latest versions of such technologies via simple web queries.

A web service also provides an architecture that will allow human language technologies (HLTs) to be integrated into larger software systems. By adopting a service-orientated architecture, existing resources and tools can also be used to develop complex component-based systems (Boehlke, 2010). Several such systems already exist in Europe and the United States, for example Stanford CoreNLP[1] (Manning *et al.*, 2014), Aylien[2], Weblicht[3] (Hinrichs *et al.*, 2010), and Tanl Pipeline[4] (Attardi *et al.*, 2010). etc. Furthermore, web services can be updated relatively quickly, allowing users to get the latest version of the technologies at all times.

In this paper, we describe a project where 61 existing text-based core technologies were ported to Java-based web services from various architectures. The first part of this paper provides a brief background and details on the relevant languages the technologies were developed for. This is followed by a short description of three previous projects in which the technologies were developed, as well as a description of the technologies themselves. We then describe the API and a simple web front-end capable of completing various predefined tasks in the following sections. We conclude with some information on a current project and future considerations.

2 Background

The South African community, with its rich diversity of 11 official languages, is an emerging market where the development of language resources and HLTs contribute to the promotion of multilingualism and language development. The development of language resources for the official languages contributes significantly to bridging the divide between the privileged and the marginalised in terms of access to information.

There are 11 official languages in South Africa, generally categorised into five language family groups. The conjunctively written Nguni languages include isiZulu (ZU), isiXhosa (XH), isiNdebele (NR), and SiSwati (SS). The disjunctively written languages include the Sotho languages Sesotho (ST), Setswana (TN), Sesotho sa Leboa (NSO), and Tshivenḓa (VE) and the disjunctively written Tswa-Ronga language, Xitsonga (TS). Finally, there are two Germanic languages, English (EN) and Afrikaans (AF)

[1] http://nlp.stanford.edu:8080/corenlp/process
[2] http://aylien.com/
[3] http://weblicht.sfs.uni-tuebingen.de/weblichtwiki/
[4] http://tanl.di.unipi.it/en/api

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics-System Demonstrations, pages 43–49
Melbourne, Australia, July 15 - 20, 2018.

(Prinsloo & de Schryver, 2002). Apart from English, all South African languages are considered resource-scarce with relatively little data that can be used to develop NLP applications and technologies.

Over the past two decades, the South African government has continuously supported HLT related text and speech projects. These projects have generated NLP resources in the form of data, core technologies, applications and systems that are immensely valuable for the future development of the official South African languages. Although these resources can be obtained in a timely fashion from the Language Resource Management Agency of the South African Centre for Digital Language Resources[5] (SADiLaR), access to these resources can still be considered limited, in the sense that technically proficient persons or organisations are required to utilise these technologies. One way to improve access to these technologies is to make them available as web services. At the Centre for Text Technology, we previously developed freely available web services for machine translation between several South African language pairs[6], and build on this experience to develop the web services.

The web services described in this paper entails the implementation of existing technologies as web services that are accessible via an application programming interface (API) and a user-friendly web application which leverages the API, described in Section 5. These services can process word lists, running text, documents or scanned images as input. The following section provides a brief overview of the individual technologies that have been implemented in the API.

3 Technologies

All the technologies included in the web services were developed over a period of eight years through three projects, NCHLT Text: Phase I, II and III. These projects were initiated and funded by the National Centre for Human Language Technology (NCHLT) of the Department of Arts and Culture (South African government). The technologies and resources described below were only developed for 10 of the South African languages, since there are well known and readily available text-based technologies for English, such as the Stanford CoreNLP, that can be used on South African English. The three projects and the resulting technologies of each, are briefly described in the following subsections.

3.1 NCHLT Text: Phase I

The first phase of the NCHLT Text project focussed on establishing the foundational resources and technologies for further development of the NLP industry in South Africa. For each language, text corpora from government domain sources were collected to develop a one-million-word corpus for each language. From these corpora, language experts for each of the 10 languages annotated 50,000 tokens per language (and an additional 5,000 tokens for testing) on three levels, namely part of speech (POS), lemma, and morphological composition. In addition to the annotated corpora, five core technologies were developed for each language. These technologies were sentence separators, tokenisers, lemmatisers, morphological decomposers, and POS taggers. Brief descriptions of each technology developed during this phase of the project and ported to web services, are provided below. More detailed descriptions of the technologies are available in Eiselen and Puttkammer (2014).

Sentence separation is a pre-processing step for tokenisation in a typical NLP pipeline. The sentence separators developed during this project are rule-based and split sentences based on language specific characteristics, to ensure that abbreviations and numbering correctly remain part of different sentences.

The tokenisers are also language-specific, rule-based technologies that split sentences into tokens, typically words and punctuation, and are a necessary pre-process for all other NLP tasks.

The POS taggers developed during the project were trained on the 50,000 POS annotated data tokens developed in the project. The implementation uses the open source Hidden Markov Model (HMM) tagger, HunPos (Halácsy *et al.*, 2007). Since HunPos is not a Java-compliant library, it was necessary to port the POS taggers to a Java library, nlp4j[7].

For the initial development and release of the web services, the lemmatisers and morphological decomposers were not included as they are rule-based technologies, with more than 150 rules

[5] http://repo.sadilar.org/handle/20.500.12185/7

[6] https://mt.nwu.ac.za/

[7] https://emorynlp.github.io/nlp4j/

each. See Section 7 for more detail on a current project tasked with additional annotation in order to develop machine learning-based technologies.

3.2 NCHLT Text: Phase II

Building on the resources created during the first NCHLT Text project, the second phase focussed on named entity recognition, phrase chunking and language identification. Named entity recognisers and phrase chunkers were developed from an additional 15,000 tokens per language annotated during the project. The language identifier (LID), which was developed to classify text as one of the 11 official languages, was trained on the text corpora collected during the first NCHLT Text project along with an English corpus also collected from government domain sources.

The named entity recognisers were trained using linear-chain conditional random fields (CRF) with L2 regularisation. See Eiselen (2016a) for details on development, evaluation, and accuracy.

The phrase chunkers were also trained with linear-chain CRFs from annotated data, and additionally use the POS tags as a feature by employing the previously developed POS taggers. Eiselen (2016b) provides the full details on development, evaluation, and accuracy of the phrase chunkers.

Both the named entity recognition and phrase chunking core technologies were implemented in the web services using the CRF++[8] Java library.

LID employs character level n-gram language models (n=6) and measures the Euclidean distance between the relative frequencies of a test model and all language models, selecting the one with the lowest distance as the probable language. In the web services, LID is performed on line level, and returns the probable language for each line in the input text. The first version of the LID was implemented in Python, and the web services version was implemented in Java. Evaluation results and implementation details are available in Hocking (2014).

3.3 NCHLT Text: Phase III

The third phase of the NCHLT Text project saw the development of Optical Character Recognition (OCR) models as well as improving access to all the technologies through the development of the web services.

The OCR models for the South African languages were developed using Tesseract[9] and accommodate the diacritic characters required for four of the South African languages. See Hocking and Puttkammer (2016) for the development and evaluation results of these OCR models. For the implementation of OCR in the web services, tess4j[10] was used.

4 Implementation

The web services are implemented as a simple three-tiered Java application, consisting of the API, a Core Technology Manager (Manager for the remainder of the paper) and the individual core technology modules.

The API is responsible for handling all incoming requests, validating parameters and headers, sending parameter data to the Manager for processing and for relaying processing results back to the requestor. The Manager is responsible for initialising and loading the technologies, processing the data from the API, and sending the result back to the API. The core technology modules process the input data and perform the various required analyses. Each of these tiers are described in more detail below.

4.1 NCHLT web services API

The API is a RESTful web service that is both maintainable and scalable. The service is based on the Jersey framework[11], as it is an open source, production quality framework for developing RESTful services in Java. The API is also designed in such a way that new language and technologies can be added at any point without affecting existing API calls. The API uses an authentication process providing restricted access to the available services of the API. The authentication process uses token-based authentication and provides the requestor with a session token that gives the requestor permission to access any future requests made to the API until the requestor's session expires. The access to the list of languages and technologies requests is not protected by the authentication process, and is therefore open to use without obtaining a session token. The API also allows the requestor to request the progress of a

[8] https://github.com/taku910/crfpp

[9] https://github.com/tesseract-ocr/

[10] http://tess4j.sourceforge.net

[11] https://jersey.github.io/

technology that is being used to process the requestor's data.

Four functions are supported by the API, which can be accessed by either GET or PUT calls, depending on whether a string of text or a file is sent for processing. The first two calls do not require authentication as described above, and return either the set of languages that are supported for a particular core technology, or a list of core technologies that are supported for a particular language. These two functions ensure that callers can correctly access those technologies that are available for particular languages.

The two functions that require authentication are the call to a specific core technology, and the progress call, which provides progress information on a user's call to a specific technology.

Most of the technologies available via the API require a language parameter in the form of an ISO-639 abbreviation of two or three letters, and some form of textual input in the form of either a list, running text or a file. The OCR module does require a language to be specified, but can only process image files in one of the standard image formats (.png, .jpg, .tiff, or .pdf), while LID only needs text or a file as it returns the language for each line in the input.

The API is called using a GET call[12] and should always consist of the following information:

- the server (and optional port number) on which the service is being hosted;
- the technology, either by number or shortened name;
- the ISO-639 two-letter language code;
- Unicode text that should be processed by the technology; and
- the authentication token included in the request header as the authToken property.

Upon receiving a request, the API validates the parameters and the session token to ensure that all the information needed to use the relevant technology is present. If the request passes the validation, the input and language information is submitted to the Manager that handles the initialisation of the requested core technology module. The Manager then validates the parameter data once again, sends the data for processing by the relevant core technology and returns the result back to the API.

[12] http://{server:port}/CTexTWebAPI/services?core={technology}&lang={code}&text={text}

4.2 Core technology manager

The Manager is tasked with handling the different core technology modules that are loaded for different languages across one or more threads or servers. The Manager controls this by keeping a register of all the modules that have been launched, as well as progress information to determine whether any given module is available for processing when a new request is received from the API. Technologies are loaded in memory as they are requested by the Manager. This allows the technologies to process the data more efficiently and in effect improves the response times to the requestor. Since many of the modules loaded by the Manager require relatively large statistical models to process data, and many of the modules are reused in several of the module pipelines, modules are not immediately discarded. Rather than destroying the loaded module, it is kept in memory to be available for a new call, which significantly reduces the processing time, since it is not necessary to reload the module or its underlying models for each new API call.

In addition to managing the individual modules that are loaded at any given time, the Manager also manages shared tasks, such as file handles and error handling, which can be reused by any of the core technology modules as necessary. This simply ensures that all file upload and download procedures are managed in a consistent, reusable fashion. Finally, it is also important to note that all the modules reuse models and attributes that are shared between multiple instances of the class and are thread-safe. Consequently, running multiple instances simultaneously does not cause any information corruption, race conditions, or related multithreading problems, while limiting the load time and memory required to process data.

4.3 Core technology modules

As mentioned earlier, the development of the web services focussed on transferring existing linguistic core technologies for South African languages to a shared code base that was accessible via a RESTful API. Over the course of the previous projects, various developers used different underlying technologies and programming languages to implement the core technologies. During this project, it was decided to consolidate these disparate technologies into a single code base, with various shared components that will

make maintenance and updates of these technologies significantly more efficient.

During the design phase it was decided to port all core technologies to Java, for three reasons. First, Java is supported across most operating systems, allowing the deployment of the technologies and services across many different architectures. Second, Java provides a wide array of freely available and well tested libraries to facilitate the development and distribution of the technologies and web services. A third factor that was taken into consideration is that the core technology modules developed for the web service could also be reused in other user-facing applications, specifically an offline corpus search and processing environment developed in parallel to the web services, CTexTools, version 2[13]. To facilitate distributed computing across multiple servers, each of the core technology modules are also implemented as servlets, which can be initialised by the manager. This allows for multiple versions of the same technology to be run on multiple threads and servers as necessary.

Although the primary focus of transferring the modules was for inclusion in the web services, this transfer also allowed for better integration between the different modules that have been developed at the Centre for Text Technology. All the transferred modules are based on a shared interface class, ICoreTechnology, which in turn implements a shared abstract class CoreTechnology. These are relatively simple shared classes, but have the significant benefit that all the core technologies can be called and handled by the Manager in a systematic, consequent manner. This in turn means that adding technologies to the set of available modules is relatively straightforward, and would immediately iterate through the rest of the API architecture, without requiring updates to the API or Manager itself.

Another consideration in the transfer of the technologies to a shared code base, is the fact that most of the technologies have an interdependence, typically forming pipelines that are required to process a string. As an example, the phrase chunker for a particular language is dependent on the output of the POS tagger as one of its features. The POS tagger in turn is dependent on tokenisation for that language, and tokenisation is dependent on sentence separation to complete its processing. This means that for phrase chunking to be performed, first sentence separation must be performed, then tokenisation, then POS tagging, and only then can the feature set be created for the string that must be phrase chunked. In the current architecture, this entire chain is inherently implemented, and the phrase chunker only needs to call the POS tagging module for the specific language, which then in turn calls the module(s) that are necessary for tagging to be performed. See Figure 1.

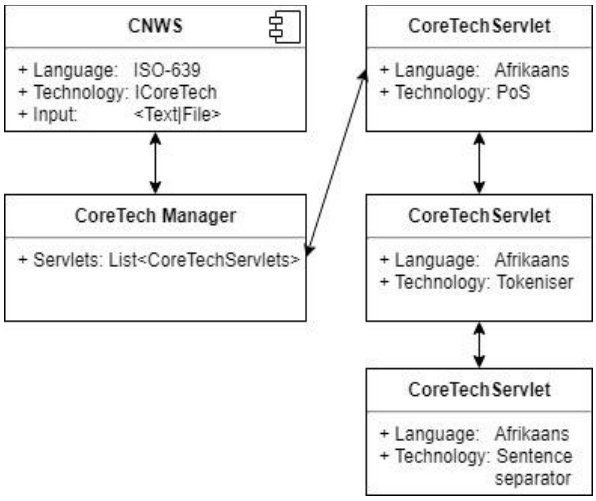

Figure 1: Example of system workflow

The modules required for each technology module are entirely handled by the Manager, which means that core technologies that are typically used in most modules, such as tokenisation, can effectively be reused by various instances of modules that require the shared module.

Due to several factors, the web services are currently only deployed on a single 12 core virtual server with 32Gb memory. In order to test the reliability of the technologies and the responsiveness of the service, a set of load tests were performed on the web services, simulating 70 users simultaneously processing text files of approximately 100,000 tokens, with different technologies in different languages. The entire scenario of processing the approximately 7 million tokens completes within 10 minutes, equating to a processing rate of around 11,700 tokens per second. In a secondary test on the slowest of the technologies, i.e. named entity recognition, for 10 concurrent users, each processing 100,000 words, the service completes in 3.5 minutes, for a rate of 1,400 tokens per second. This is primarily due to the fact that named entity recognition uses the most intricate pipeline, including tokenisation,

[13] https://hdl.handle.net/20.500.12185/480

sentence separation, part of speech tagging, and extended feature extraction.

5 Web application

To make the technologies developed during the various phases of the NCHLT project more accessible, a simple web application was also created. This application specifically aims to accommodate users who are unaccustomed to service-orientated architectures, and for whom using these types of architectures can be quite challenging. As such, it was prudent to develop a basic interface to assist users in using the services to complete certain tasks. Thus, we developed a web-based, user-friendly graphical user interface capable of completing various tasks by providing predefined chains of the web services detailed above. For example, if a user needs to perform POS tagging on a document, the user can upload the document and select POS tagging and the relevant language. The system will automatically perform tokenisation and sentence separation before using the POS tagging service to tag the user's document. To facilitate easy and quick processing, a user can provide text, select the required options, process the text, and view or download the results. Detailed documentation on using the API, as well as the web application, is also provided. The tag sets used for all annotation are provided in the help page. The web application is available at http://hlt.nwu.ac.za/.

6 Conclusion and future work

In this paper, we provided an overview of a new web service and application that provides access to 61 different text technologies for South African languages. This implementation allows any developer to access and integrate one of these language technologies in their own environment, while ensuring that the latest versions of these technologies are used at any time. Finally, a simple, user-friendly, web application was described that provides access to predefined chains of NLP technologies for use by end-users who are not as technically proficient, but can use the technologies in their own research work.

Given the flexible nature of the web services and underlying infrastructure, it is foreseen that other language technologies will be included in the service as they become available. The South African government also recently established SADiLaR, a national research infrastructure focussing on the development and distribution of linguistic and natural language processing resources.

There is currently a project underway to extend the set of annotated text corpora from 50,000 to approximately 100,000 tokens. These extended annotated data sets could then be used to create improved core technologies for the South African languages.

7 Acknowledgements

The NCHLT web services project was funded by the Department of Arts and Culture, Government of South Africa. The expansion of the annotated data is funded as a specialisation project by SADiLaR.

References

Alonso, G., Casati, F., Kuno, H. & Machiraju, V. 2004. Web services. Springer-Verlag, Berlin.

Attardi, G., Dei Rossi, S. & Simi, M. 2010. The TANL Pipeline. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC 2010): Workshop on Web Services and Processing Pipelines in HLT*. European Language Resources Association (ELRA), p. 15-21.

Boehlke, V. 2010. A generic chaining algorithm for NLP webservices. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC 2010): Workshop on Web Services and Processing Pipelines in HLT*. European Language Resources Association (ELRA), p. 30-36.

Eiselen, R. 2016a. Government Domain Named Entity Recognition for South African Languages. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. European Language Resources Association (ELRA), p. 3344-3348.

Eiselen, R. 2016b. South African Language Resources: Phrase Chunking. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. European Language Resources Association (ELRA), p. 689-693.

Eiselen, R. & Puttkammer, M.J. 2014. Developing Text Resources for Ten South African Languages. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC 2014)*. European Language Resources Association (ELRA), p. 3698-3703.

Halácsy, P., Kornai, A. & Oravecz, C. 2007. HunPos: an open source trigram tagger. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, Companion Volume: Proceedings of the Demo and Poster Sessions*. Association for Computational Linguistics, p. 209-212.

Hinrichs, E., Hinrichs, M. & Zastrow, T. 2010. WebLicht: Web-based LRT services for German. In *Proceedings of 48th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Association for Computational Linguistics, p. 25-29.

Hocking, J. 2014. Language identification for South African languages. In *Proceedings of the Annual Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech): Poster session*. Pattern Recognition Association of South Africa, p. 307.

Hocking, J. & Puttkammer, M.J. 2016. Optical character recognition for South African languages. In *Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech), 2016*. IEEE, p. 1-5.

Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. & Mcclosky, D. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. The Association for Computational Linguistics, p. 55-60.

Prinsloo, D. & De Schryver, G.-M. 2002. Towards an 11 x 11 array for the degree of conjunctivism/disjunctivism of the South African languages. *Nordic Journal of African Studies*, 11(2):249-265.