This document is part of the Coordination and Support Action "Preparation and Launch of a Large-scale Action for Quality Translation Technology (QTLaunchPad)". This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no. 296347.

Deliverable D 3.3.1

Adapted tools for the QTLaunchPad infrastructure

Author(s): Dimitris Galanis, Stelios Piperidis, Juli Bakagianni, Sokratis Sofianopoulos

Dissemination Level: Public

Date: 27.11.2014

Grant agreement no.	296347	
Project acronym	QTLaunchPad	
Project full title	Preparation and Launch of a Large-scale Action for Quality Translation Technology	
Funding scheme	Coordination and Support Action	
Coordinator	Prof. Hans Uszkoreit (DFKI)	
Start date, duration	1 July 2012, 24 months	
Distribution	Public	
Contractual date of delivery	June 2014	
Actual date of delivery	27 November 2014	
Deliverable number	3.3.1	
Deliverable title	Adapted tools for the QTLaunchPad infrastructure	
Type	Prototype	
Status and version	Final, v2.0	
Number of pages	18	
Contributing partners	ILSP, DFKI, DCU	
WP leader	DFKI	
Task leader	ILSP	
Authors	Dimitris Galanis, Stelios Piperidis, Juli Bakagianni, Sokratis Sofianopoulos	
EC project officer	Aleksandra Wesolowska	
The partners in QTLaunchPad are:	Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany	
	Dublin City University (DCU), Ireland	
	Institute for Language and Speech Processing, R.C. “Athena” (ILSP/ATHENA RC), Greece	
	The University of Sheffield (USFD), United Kingdom	

Version History

Version	Date	Comments
1.0	27/06/2014	
2.0	27/11/2014	With respect to the comments in the final review report, the following items have been included: • information about scalability of language processing tools

For copies of reports, updates on project activities and other QTLaunchPad-related information, contact:

DFKI GmbH
QTLaunchPad
Dr. Aljoscha Burchardt aljoscha.burchardt@dfki.de
Alt-Moabit 91c Phone: +49 (30) 23895-1838
10559 Berlin, Germany Fax: +49 (30) 23895-1810

Copies of reports and other material can also be accessed via http://www.qt21.eu/launchpad

Table of Contents

1 Executive Summary

This report briefly describes the language processing layer of the QT21 repository. The language processing layer integrates and links to a growing number of services performing: a) tokenisation, sentence splitting, pos tagging, lemmatisation, chunking, dependency parsing, and named entity recognition for monolingual and multilingual textual datasets, and b) text alignment for multilingual textual datasets. These services are currently offered for English, German, Portuguese and Greek. They can be locally or remotely deployed and chained into appropriately defined, internally interoperable workflows.

2 Introduction

The data infrastructure and data preparation (pre-processing) workflows supported by QTLaunchPad are based on and extend the META-SHARE infrastructure[1]. The META-SHARE functionalities have been extended, at a prototype level, by the provision of an additional language processing mechanism for processing language datasets (essentially language corpora) with appropriate natural language tools. Language processing tools are documented with the appropriate metadata in the QT21 META-SHARE compliant repository[2] and are provided as web services through the language processing layer. When the user selects to process a dataset, a list of all available annotation services for each relevant annotation level (e.g. tokenization & sentence splitting, POS tagging, lemmatization, alignment) are provided for the given language, and resource type. As soon as the user selects a tool, the server invokes a service that dispatches the corpus to the specific web service for processing. The system informs the user about the requested job via the messaging service of the platform. When the processing has been completed, the new (annotated) dataset is automatically stored and indexed in the repository, and the user is appropriately informed. If the user, for any reason, requests to process a dataset with a specific tool, and this dataset has already been processed by the specific tool, then the system will just forward the user to the processed dataset that has been created and stored in the repository.

[1] www.meta-share.eu, www.meta-share.org
[2] http://qt21.metashare.ilsp.gr/

3 Overview of the design

As said in the introduction, the existing META-SHARE infrastructure has been extended so that it can integrate and provide language, essentially text, processing services and workflows for monolingual and bilingual resources for the English, German, Portuguese and Greek, languages (as defined in the project's DoW). The respective language processing tools are documented with the appropriate metadata in the QT21 META-SHARE compliant repository2 and are provided as web services through the language processing layer. In a typical scenario, an authenticated user selects via the QT21 repository web-based user interface a language resource and the atomic or composite service (a.k.a. workflow) (s)he wishes to use to process the resource. The QT21 application sends to the language processing (LP) layer a processing request via a standard web service (REST) and waits until it is completed. When LP gets the request, it starts to process the specified resource and when it finishes it notifies the QT21 application so that the result of the processing is added in the QT21 repository. Finally, the QT21 application sends the user an email with the link to the newly created resource. In the following two sections we present the adaptations that were made in META-SHARE and we give an overview of LP application.

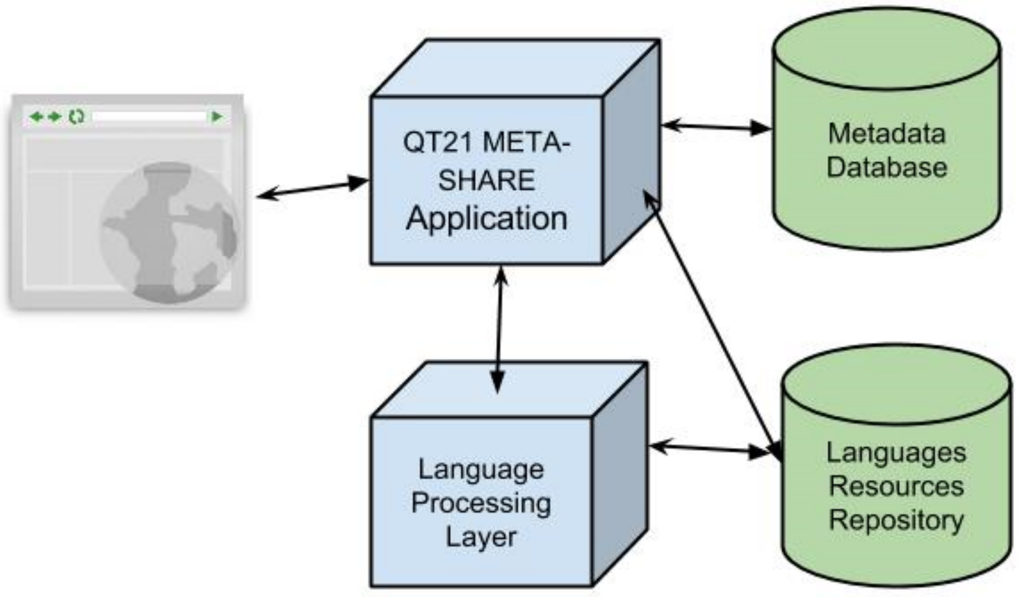
QT21 META-SHARE Application
Metadata Database
Language Processing Layer
Languages Resources Repository

Figure 1: QT21 repository architecture

4 META-SHARE extensions

The main adaptations and extensions of the existing META-SHARE infrastructure, in view of integrating a language processing layer, concern the metadata model of it for describing and documenting the properties of datasets and language processing services, as well mechanisms for automatically creating the metadata records of the newly generated resources as a result of processing using an annotation service or workflow.

In addition certain extensions concern the providers of Language Resources who wish to deposit their LRs to the QT21 repository and make them processable. A LR can be input to the processing mechanism offered by the QT21 repository, if it complies with the specifications described further below (and also in the respective user manual, project deliverable D3.4.3) . The provider can deposit her LR to the QT21 repository, by referring to the documentation in the provider's manual of the META-SHARE application (https://github.com/metashare/META-SHARE/tree/master/misc/docs). Additionally, the provider should upload the actual data of her LR to the QT21 repository storage. The data should be compressed and must not exceed 20MB in size. The provider must additionally select the data format of her LR. Information regarding every data format is available by clicking the help buttons of the metadata editor, which accompany each data format choice.

5 Language Processing Layer

The Language Processing (LP) layer has been implemented in Java, based on the Apache Camel framework[3]. Camel is an open-source project that provides libraries which enable the easy integration of different components and technologies and the creation of (data) processing workflows. Given that the services deployed in QTLaunchPad are atomic and composite (workflows) language processing services which use several different technologies (e.g. UIMA, GATE, SOAP, OpenNLP), Apache Camel was a natural choice. The implemented LP is bundled as a web application and can be deployed in a standard java-based web container. In our tests, we

[3] http://camel.apache.org/

used JETTY, which is small, fast and embeddable server that powers many successful projects (e.g. Solr).

6 Processing

LP's workflows are implemented based on a variety of natural language processing services. These services run either locally (within our application) (LOC), or they are accessed via remote services (RMT). In the following table, we present the processing services we use for each language, as well as how each tool is running (LOC or RMT).

Language	**NLP tool**	**Running mode**
Greek	ILSP Tokenizer/Sentence Splitter	UIMA RMT
Greek	ILSP POS Tagging	UIMA RMT
Greek	ILSP Lemmatizer	UIMA RMT
Greek	ILSP Dependency parser	UIMA RMT
Greek	ILSP GATE Named Entity Recognition	GATE LOC
English	OpenNLP Sentence Splitter	OpenNLP LOC
English	OpenNLP Tokenizer	OpenNLP LOC
English	OpenNLP POS Tagger	OpenNLP LOC
English	OpenNLP NP chunker	OpenNLP LOC
English	OpenNLP NERC	OpenNLP LOC
English	DCU Tokenizer	SoapLab RMT
English	DCU Sentence Splitter	SoapLab RMT
English	DCU POS tagger (Berkeley tagger)	SoapLab RMT
English	DCU Lemmatizer and Tagger	SoapLab RMT
Portuguese	OpenNLP Sentence Splitter	OpenNLP LOC
Portuguese	OpenNLP Tokenizer	OpenNLP LOC
Portuguese	OpenNLP POS Tagger	OpenNLP LOC
Portuguese	Lisbon NLX Sentence Splitter[4]	SOAP RMT
Portuguese	Lisbon NLX Tokenizer	SOAP RMT
Portuguese	Lisbon NLX POS tagger	SOAP RMT

[4] We are grateful to the NLX-Natural Language and Speech Group of the University of Lisbon and its head Antonio Branco for kindly providing access to the LX-Center services.

German	OpenNLP Sentence Splitter	OpenNLP LOC
German	OpenNLP Tokenizer	OpenNLP LOC
German	OpenNLP POS Tagger	OpenNLP LOC
German	Heart of Gold Sentence Splitter and Tokenizer (DFKI)	SOAP RMT
German	Heart of Gold POS Tagger (DFKI)	SOAP RMT
German	Heart of Gold Morphological Analyzer (DFKI)	SOAP RMT
German	Heart of Gold Named Entity Recognizer (DFKI)	SOAP RMT
Bilingual pairs of the languages above	Hunalign sentence aligner	SoapLab RMT or LOC if we deploy it locally

Table 1: NLP services integrated in the QT21 language processing layer

Each set of workflows, for example the UIMA-based ones for the Greek language (see Figure 2), can be modelled as an acyclic directed graph (tree) where each node corresponds to a processing service. The processing of a data chunk is performed by following a path in such a workflow tree. For example, in the case where the input is a raw text the starting point is the root of the tree (e.g. splitter/tokenizer in Figure 3.). However, LP is also capable of processing already annotated resources (e.g. a POS tagged text can be lemmatized and parsed for dependencies by following the appropriate path).

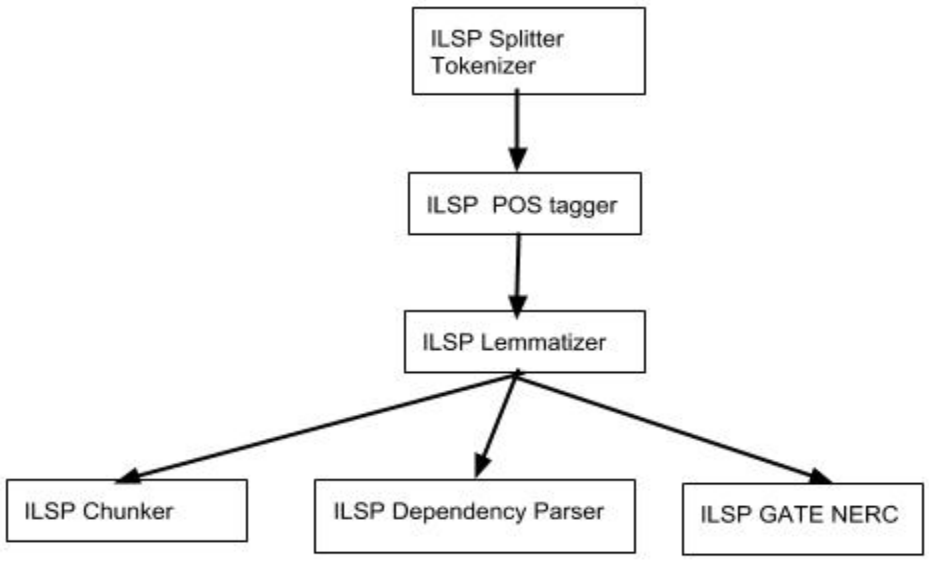

Figure 2: Workflow tree for the Greek Language.

6.1 Input

Currently, LP implements services and workflows that can process a) monolingual resources in raw text as well as XCES format and b) bilingual resources in TMX, MOSES, and XCES formats. The resources are stored in the QT21 repository in a compressed format (e.g. .zip, tar.gz, .gz). Initially, a service or a workflow decompresses the specified resource file and then uses an appropriate reader that splits the content of the extracted files in smaller text (data) chunks, so that we comply with any file size limitations that a service might have. These chunks are then forwarded to the appropriate NLP service/workflow. A symmetric workflow that collects the data chunks after the processing and merges them in a single compressed resource is initiated as soon as the service/workflow has completed the data processing.

6.2 Output

The output format for the produced annotations of the LP is pertinent to the specific workflow that is executed. For example, the annotations of a UIMA-based pipeline are serialized in the XML Metadata Interchange format (XMI) whereas in OpenNLP and DCU workflows the output is stored in a custom XML representation and in a text-based format, respectively. However, in all cases, the files with the annotations which are produced from the processing of a language resource are checked for potential software errors (e.g. number of files processed as expected), archived in a single file and added to the QT21 repository.

6.3 Error Handling

In the cases where a tool is remotely accessed (RMT) we use a timeout mechanism to prevent the application from hanging. When such a timeout occurs (for any reason), the processing of the corresponding data chunk is terminated, i.e. it is not forwarded to the next step of the workflow. The same policy is also applied in case an error occurs in a service that runs locally (LOC).

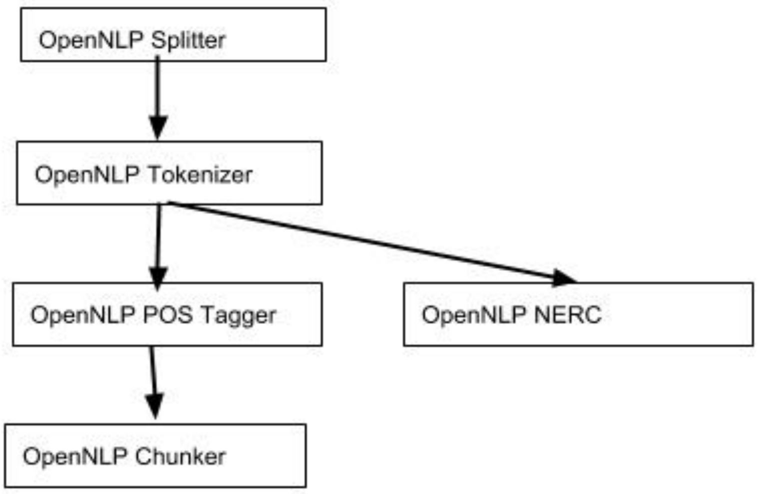

Figure 3: Workflow tree for the English OpenNLP tools.

6.4 Concurrency

To ensure that the LP system concurrently executes several workflows or parts of a workflow, we use data queues and multiple threads. In particular, there are separate threads for a) decompressing an input resource, b) reading and splitting the input files, and c) for archiving the output files. Moreover, there are one or more threads for executing a processing pipeline depending on the workflow. For example, in the case of UIMA-based ones there is a pool of N threads per language processing tool that send requests and receive results from the services.

7 Workflow assumptions and limitations

Currently, each QT21 NLP workflow chains together components or services of the same suite/family of tools, for example OpenNLP or the PANACEA-DCU services. To accommodate cases where the services deployed belong to different suites, we have developed the appropriate converters. For example, in the UIMA-based tree of Figure 2, where a GATE-based Named Entity Recognizer is integrated in the respective Named Entity Recognition workflow, the UIMA output of the processing services preceding named entity recognition is converted to the GATE format and is fed to the GATE-compatible Named Entity Recognizer (e.g. Tokenizer->Splitter->POSTagger->UIMA-GATE Converter->NERC).

For adding new atomic or composite, interoperable integrated services (SOAP, Open NLP, UIMA, GATE compatible) the provider has to appropriately document his/her services/workflows using the QT21 metadata schema, i.e. at least service name and location, platform and service version, languages, acceptable input/output formats, while the LP has to be appropriately enhanced with a mechanism that dynamically creates the appropriate workflows based on the given description.

Enabling the user to define and deploy custom workflows, cross-suite or not, is on our agenda for the immediate future. The implementation of cross-suite workflows requires the development of several data format converters for each pair of different technologies (e.g. UIMA-GATE, DCU-OpenNLP). Understandably, there are several performance, compatibility and above all interoperability issues that arise in such cases and have to be investigated and addressed. For example, the machine learning models on which the tools are based, have been trained and optimized using the output of a specific tool. This fact, may lead to significant drop of the performance of a tool. Moreover, in a mixed workflow it is possible, not to say highly expected, that a tool may use a different tagset than the one required as input in the next step. This may lead to either a software error or to an output of low quality.

In addition, worthwhile noting is the dependence of the LP application on the design properties of individual, remotely called workflows. Such properties may constrain the amount of data a workflow can process per unit time (workload), as some workflows are:

a) deployed in a single machine which might also have limited processing and memory capabilities
b) inherently designed and implemented to serve only non-concurrent requests.

7.1 Licensing issues

Last, but not least, considering the QT21 repository operations from the legal framework point of view, we have adopted a rather simplified operational model by which only openly licensed, with no no-derivatives (ND) restriction, datasets can be processed by openly licensed services and workflows. In future versions, QT21 will be equipped with a business logic that will allow processing of otherwise licensed datasets and services supporting the appropriate business models.

7.2 Scalability of the prototype implementation

This section reports on the scalability tests that were performed on the QT21 META-SHARE processing layer, following the recommendations of the reviewers[5]. All tests were performed on a QT21 language processing application that has been deployed in a single machine, an Ubuntu Virtual server with 8 cores (2 Intel Xeon E5-2430 @2.2GHz) and 8Gb of RAM. The processing application was invoked either via QT21 META-SHARE web application or through a simple test client application that we have developed. For the tests, language resources of five different sizes stored in the QT21 repository were sent to various processing workflows. These workflows were divided into two categories, local and remote, depending on whether the service runs locally on processing application server, or remotely invoked via web services. For local services, the resources used for the tests were of 1MB, 10MB and 50MB, while for the remote ones the corresponding sizes were 500KB, 5MB and 10MB. Remote services were tested with smaller resources since there are currently running on no high-end machines. Table 2 presents the results of the tests for the local services, while Table 3 the results for the remote ones.

[5] The QT21 METASHARE repository server has been already thoroughly tested during the METASHARE project

Local Service Name	Requires	Lang-uage	Processing time (hh:mm:ss)		
			Small-sized resources (~ 1MB)	Medium-sized resources (~ 10 MB)	Large-sized resources (~ 50 MB)
ILSP Tokenizer and Sentence Splitter[6]	-	EL	0:01:32	0:02:37	0:06:24
ILSP POS Tagger	ILSP Tokenizer and Sentence Splitter	EL	0:02:02	0:10:42	0:45:46
ILSP POS Tagger and Lemmatizer	ILSP Tokenizer and Sentence splitter	EL	0:03:05	0:18:16	1:22:05
ILSP Dependency Parser	ILSP POS Tagger and Lemmatizer	EL	0:03:07	0:17:51	1:21:10
ILSP GATE-based Named Entity Recognizer	ILSP POS Tagger and Lemmatizer	EL	0:03:05	0:18:05	1:22:22
OpenNLP Sentence Splitter	-	EN	0:01:32	0:01:32	0:01:38
OpenNLP Tokenizer	OpenNLP Sentence Splitter	EN	0:01:32	0:02:04	0:04:16

[6] The ILSP language processing web-services are actually permanently running on a virtual machine hosted by the Greek Cloud Infrastructure ~okeanos, therefore locality of processing is slightly relativized in the context of this table. Since the management of these services is undertaken by ILSP, we had the opportunity to stress the offered services with data of the same size as the ones we used for testing the scalability of actual local services.

OpenNLP POS Tagger	OpenNLP Tokenizer	EN	0:01:32	0:03:35	0:12:51
OpenNLP NP chunker	OpenNLP POS Tagger	EN	0:02:02	0:07:37	0:32:34
OpenNLP Named Entity Recognition	OpenNLP Tokenizer	EN	0:02:02	0:07:07	0:29:52
OpenNLP Sentence Splitter	-	PT	0:01:32	0:01:32	0:01:39
OpenNLP Tokenizer	OpenNLP Sentence Split-ter	PT	0:01:32	0:03:05	0:09:17
OpenNLP POS Tagger	OpenNLP Tokenizer	PT	0:01:32	0:04:05	0:14:28
OpenNLP Sentence Splitter	-	DE	0:01:32	0:01:34	0:01:39
OpenNLP Tokenizer	OpenNLP Sentence Split-ter	DE	0:01:32	0:05:06	0:19:16
OpenNLP POS Tagger	OpenNLP Tokenizer	DE	0:02:02	0:06:06	0:24:52
Hunalign sentence aligner	-	Greek English	0:01:35	0:05:42	0:16:58

Table 2: Scalability test results for all local services

Remote Service Name	Requires	Lang-uage	Processing time (hh:mm:ss)		
			Small-sized resources (~ 500KB)	Medium-sized resources (~ 5 MB)	Large-sized resources (~ 10 MB)
DCU Tokenizer	-	EN	0:01:32	0:05:04	0:09:04
DCU Sentence Splitter	DCU Tokenizer	EN	0:01:32	0:05:34	0:09:34
TreeTagger – POS Tagger and Lemmatizer (DCU)	DCU Sentence Splitter	EN	0:16:00	2:22:38	4:40:46
Lisbon NLX Sentence Splitter	-	PT	0:03:33	0:20:34	0:39:36
Lisbon NLX Tokenizer	Lisbon NLX Sentence Splitter	PT	0:03:33	0:22:05	0:43:05
Lisbon NLX POS tagger	Lisbon NLX Tokenizer	PT	0:14:46	1:58:24	3:49:55
Heart of Gold Sentence Splitter and Tokenizer (DFKI)	-	DE	0:02:02	0:07:34	0:12:37
Heart of Gold POS Tagger (DFKI)	Heart of Gold Sentence Splitter and Tokenizer (DFKI)	DE	0:02:35	0:12:06	0:22:38
Heart of Gold Morphological Analyzer (DFKI)	Heart of Gold POS Tagger (DFKI)	DE	0:04:07	0:27:18	0:54:07
Heart of Gold Named Entity Recognizer (DFKI)	Heart of Gold POS Tagger (DFKI)	DE	0:03:35	0:25:36	0:47:08

Table 3: Scalability test results for all remote services

As shown in Tables 2 and 3 the local workflows obtain lower processing times, confirming our expectations, as the remote services were running, during this scalability testing, on no high-end machines and were also of course subject to network delays and congestions.

In Table 2, we notice that the processing times for all resource sizes of either the ILSP GATE-based Named Entity Recognizer or the ILSP Dependency Parser workflow are almost the same as the ones of the ILSP POS Tagger and Lemmatizer even though the latter is required in both of them. This is due to the fact that the three aforementioned tools run in parallel and the processing time of the ILSP POS Tagger and Lemmatizer workflow is longer, therefore being the bottleneck of the whole process. The time differences that occur are mainly due to a) network delays that in some cases may trigger timeouts b) the CPU usage/availability of the server that hosts the ILSP services and which is shared by more than one VMs.

Service Name	Lang-uage	Resource size	Processing time (hh:mm:ss)	Processing time (sec)
OpenNLP POS Tagger	Portuguese	~50MB	0:16:54	1014
OpenNLP POS Tagger	German	~50MB	0:27:21	1641
OpenNLP Named Entity Recognition	English	~50MB	0:24:29	1469
ILSP GATE-based Named Entity Recognizer	Greek	~50MB	1:22:20	4940

Table 4: Processing times of four local services when called in parallel

Table 4 presents the processing times of four workflows that were started at the same time and were executed concurrently. Each of them is comparable (almost the same) with the corresponding time that it takes to the same workflow to process the same resource in the same server without other concurrent workflow executions (see Table 2).

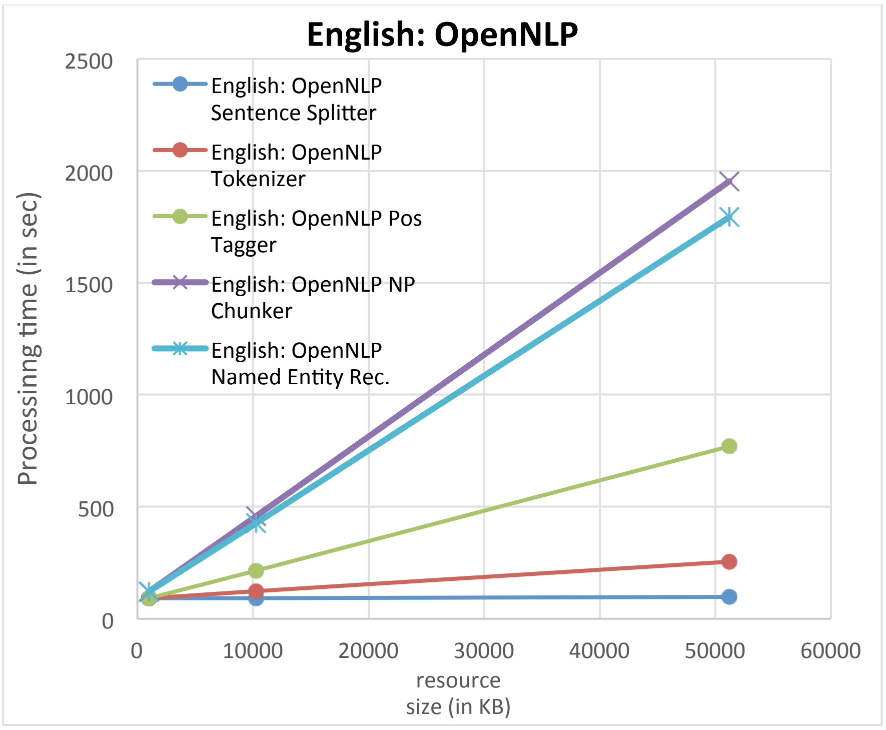

Figure 4: Plot of processing times over resource size for all local English services

As shown in Figure 4, the processing times of all OpenNLP services grow linearly with resource size thus excluding any case of memory leakage during operation. The same conclusion is also drawn from the plots for all other local and remote services.