OpeNER and PANACEA: Web Services for the CLARIN Research Infrastructure

Davide Albanesi
Istituto di Linguistica Computazionale
"A. Zampolli" (CNR-ILC)
Via G. Moruzzi, 1 - 56124 Pisa (ITALY)

Riccardo Del Gratta
Istituto di Linguistica Computazionale
"A. Zampolli" (CNR-ILC)
Via G. Moruzzi, 1 - 56124 Pisa (ITALY)

name.surname@ilc.cnr.it

Abstract

This paper describes the necessary steps for the integration of OpeNer and PANACEA Web Services within the CLARIN research infrastructure. The original Web Services are wrapped into a framework and re-implemented as REST APIs to be further exploited through both Language Resource Switchboard and WebLicht and made available for the CLARIN community.

1 Introduction and motivation

OpeNer and PANACEA[1] were two European projects funded within the 7^{th} Framework Program and covering 4 years of research initiatives on Language Resources and Technologies (LRT). OpeNer developed some Natural Language Processing (NLP) tools in order "to detect [...] entity mentions [...]", by "[...] performing sentiment analysis and opinion detection on" specific textual resources, especially in reviews for hotel accommodations and tourism at a large. Such tools were designed to be easily customizable for Academia, Research and Small and Medium Enterprises. The exhaustive list of services and lexicons developed by OpeNer as well as of the European languages covered are available at their official github.[2] The PANACEA project addressed "the most critical aspect of Machine Translation (MT): the so called language resource bottleneck." PANACEA developed a set of linguistic resources, more precisely "a 'factory' of Language Resources (LRs) in the form of a production line [...]", to automate the stages for "[...] acquisition, production, maintenance and updating of the language resources required by machine translation". The platform created in the framework of PANACEA is a virtual and distributed environment where various interoperable components can be concatenated to create specific workflows to produce several language resources in various languages. The services developed in PANACEA are of great importance for Academia, Research and Small and Medium Enterprises, especially the ones focused on MT and related technologies. OpeNer and PANACEA share many aspects: from the creation of annotated corpora and lexicons to the development of web tools and services used to analyze and build them up to the focus on specific communities. They also share the concept of interoperability as the use of the Kyoto Annotation Format (KAF) (Bosma et al., 2009) in OpeNer,[3] the Graph Annotation Format (GrAF)(Ide and Suderman, 2007) in PANACEA[4] and the Lexical Markup Framework (LMF) (**?**) in both of them states. Data and tools interoperability is a strategic goal in CLARIN. [5] And, within CLARIN, initiatives such as the Language Resource Switchboard (LRS) (Zinn, 2018) and WebLicht (Hinrichs et al., 2010) openly go towards methodologies and "systems" addressing interoperability issues between language tools and language resources. These initiatives are central in CLARIN which therefore becomes the ideal environment for the tools and Web Services offered by OpeNer and PANACEA. Lastly, the services developed within both projects and the results achieved play a key role for the CLARIN community as well. Indeed, on the one hand, the Virtual Language Observatory (VLO)[6]

[1] Respectively http://www.opener-project.eu/ and http://www.panacea-lr.eu/

[2] https://github.com/opener-project

[3] https://github.com/opener-project/kaf/wiki/KAF-structure-overview

[4] http://www.panacea-lr.eu/system/graf/graf-TO2_documentation_v1.pdf

[5] See, for instance https://www.clarin.eu/event/2019/parlaformat-workshop, https://www.clarin.eu/event/2017/clarin-workshop-towards-interoperability-lexico-semantic-resources among others.

[6] https://vlo.clarin.eu/

contains several LRs but only some specific tools for Sentiment Analysis, while, on the other hand, many LRT are available for MT. This clearly means that the latter community is already in the CLARIN community, while the former one should be helped to get more involved. It is a matter of fact that the more the tools are published and used in and through CLARIN, the more they will have an impact on the community, and this community will tend to grow. However, it is obvious that the community involvement can not be managed only from the technological point of view, but a political point of view is needed as well.

2 Current Architecture and Common Aspects

ILC4CLARIN, hosted at the National Council of Research (CNR) "Institute for Computational Linguistics A. Zampolli" in Pisa, is the first and leading CLARIN B-centre of Italian Consortium, CLARIN-IT.[7] ILC4CLARIN is already offering some of the PANACEA Web Services through a man-machine interaction available at `https://ilc4clarin.ilc.cnr.it/en/services/`, while OpeNer Web Services are offered through a local installation.

Tokenizer	http://opener.ilc4clarin.ilc.cnr.it/tokenizer	**Pos Tagger**	http://opener.ilc4clarin.ilc.cnr.it/pos-tagger
Kaf2Json	http://opener.ilc4clarin.ilc.cnr.it/kaf2json	...	...

Table 1: OpeNer Endpoints

In addition, the initial implementation of the tools has many common aspects, such as, for instance, the following:i) many tools in OpeNer and PANACEA are command line tool. To wrap command line tools, OpeNer uses Ruby[8] and builds REST Web Services, while PANACEA uses Soaplab[9] and offers SOAP Web Services; ii) OpeNer offers both POST and GET API; iii) PANACEA offers SOAP Web Services through a Web Interface; iv) simple pipelines are available in OpeNer, while a workflow engine[10] is used in PANACEA; v) Kyoto Annotation Format (KAF), Lexical Markup Framework (LMF) and Graph Annotation Format (GrAF) guarantee the interoperability among data and services at different levels. Although their architectures differ, both foster interoperability as it is shown in the Figures 1 and 2.

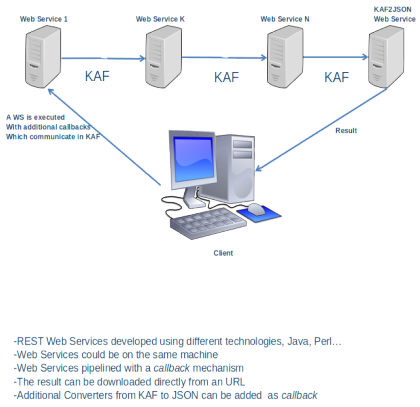

Figure 1: OpeNer Architecture

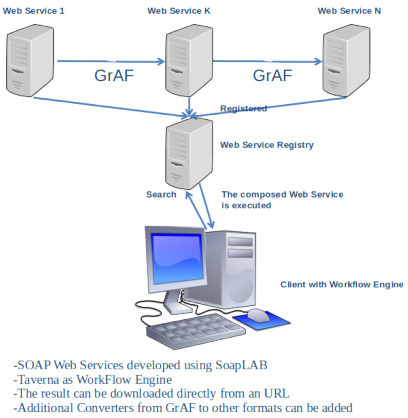

Figure 2: PANACEA Architecture

The images reported above show that the tools are ready to be inserted into workflows but, when it comes to fully satisfy the requirements of Language Resource Switchboard (LRS) and WebLicht, we need to

[7] In order `https://ilc4clarin.ilc.cnr.it/`, `http://www.ilc.cnr.it/en/`, `https://www.clarin-it.it`

[8] See `https://rubyonrails.org/`

[9] Soaplab is described at `http://soaplab.sourceforge.net/soaplab2/`

[10] See Taverna, `www.taverna.org.uk`

create new wrappers around the available tools so that we can correctly manage REST APIs that are able to "consume" and/or produce different formats. LRS does not require that the handled tools have specific output formats but, in any case, it requires the tools are able to read texts from URLs, uploaded files, or simple input boxes, while WebLicht accepts tools able to read and write the TCF format. [11]

3 Moving OpeNer and PANACEA into CLARIN

In this section we describe the strategy used to integrate OpeNer and PANACEA Web Services into the CLARIN infrastructure. This initiative is carried out by the development team at the ILC4CLARIN centre.

3.1 Technical Implementation

Firstly, we have to consider that OpeNer offers REST Web Services (by default managed by APIs), while PANACEA offers SOAP ones. This means that the former is easily wrapped into a REST context, while the latter needs to be managed with some SOAP APIs before being inserted in a REST context. We have two alternatives: either i) to use the SOAP APIs provided by SoapLab; [12] or ii) to start from the original command line programs and use a different framework to transform these scripts into REST APIs. Both alternatives have their pros and cons. The first one forces the development team to code a shell around SOAP Web Services so that they can be "executed" by a software program and not just by a web interface (as it is the current case). Fortunately, the SoapLab APIs do exactly this: they provide methods to access Web Services in a simple way, without the burden of a SOAP implementation. Anyway, they are an additional piece of software to manage: the compatibility with existing libraries, methods that become deprecated and must be replaced, security flaws ... must be addressed. The second option needed to replicate (in a sense) what was already obtained through SoapLab. For instance, a framework like CLAM[13] allows developers to transform command line programs into REST Web Services; and this would align the PANACEA and OpeNer Web Services. However, there would be a duplication in terms of both service endpoints and machines for hosting them. We simply considered this alternative non-economic (at least from our point of view), so we opted for alternative i). There are several strategies for implementing REST services; the majority follows the JAX-RS[14] specifications. We decided to use the DropWizard[15] framework which was described to the second author directly by the WebLicht developers during a hackaton[16] held in Ljubljana in 2016. This framework combines an HTTP server (Jetty), a library for JAX-RS (Jersey), a "lingua franca" (Json[17]) and many other useful development tools. Furthermore, DropWizard is very useful for decoupling the basic implementation (the actual piece of code performimg the operations) from how the code and the results of the operations are managed through the HTTP protocol. It is easy to understand that the situation of the PANACEA and OpeNer Web Services is exactly the following one: a core part (SOAP and REST Services respectively) and a wrapper that makes them accessible to programs through the HTTP protocol. In conclusion, DropWizard is a framework which helps to decouple the core tools from the corresponding web resources, as it is described in Listing 1.

[11] The TCF format is described at `https://weblicht.sfs.uni-tuebingen.de/weblichtwiki/index.php/The_TCF_Format`

[12] Java implementation in *taverna-soaplab-client* from the maven repository `http://www.mygrid.org.uk/maven/repository/`

[13] `https://proycon.github.io/clam/`

[14] `https://jcp.org/en/jsr/detail?id=339`

[15] `http://www.dropwizard.io/1.3.4/docs/`

[16] `http://www.clarin.si/info/events/mcat-workshop/`

[17] In order `https://www.eclipse.org/jetty/`, `https://jersey.github.io`, `https://www.json.org/`

```
/**
 * This is an example code which uses dropwizard, jetty and weblicht api to include
 * a core tool as a web service resource.
 ** * a lot of other imports */
import io.dropwizard.Configuration;
/**
* the resource for TCF is something as
*/
@Consumes(TEXT_TCF_XML)
@Produces(TEXT_TCF_XML)
public StreamingOutput myexample(final InputStream text) {
  OutputStream tempOutputData = null;
  .....
  /* call the core */
  calltheservice(lang, text, some_other_parameters, tempOutputData);
  .....
}
/** the core where the actual tool is executed */
public void calltheservice(String lang, Map map, InputStream i, OutputStream o) {
  MyTool tool = new MyTool();
  TextCorpusStreamed textCorpus = null;
  ....
  tool.doSomething(textCorpus);
   ...
}
```

Listing 1: DropWizard integration

In Listing 2, a Jersey client executes the OpeNer web Service at one of its endpoint (endpoints are listed in Table 1) and returns the response.

```
/* import jersey stuff */
public doSomething(...){
    client = Client.create();
    webResource = client.resource(URL_ENDPOINT);
    response = webResource.type(MediaType.APPLICATION_FORM_URLENCODED)
        .post(ClientResponse.class, formData);
    output = response.getEntity(String.class);
    setOutputStream(output);
   ....
}
```

Listing 2: Opener Snippet

```
/* import soaplab api stuff */
public doSomething(...){
  SoaplabBaseClient client = getClient(SERVICE_ENDPOINT);
  SoaplabMap results = client.
      runAndWaitFor(SoaplabMap.fromMap(getInputs()));
      .....
  Map outputs = SoaplabMap.toMap(results);
  /* manage outputs and format*/
  ....
}
```

Listing 3: Panacea Snippet

Listing 3 is quite similar to 2 but, in the latter, a client based on SoapLab APIs is responsible for executing the SOAP Web Service at the `SERVICE_ENDPOINT`. Soaplab APIs return the response that contains the analyzed text. The full code is available at `https://github.com/cnr-ilc/linguistic-tools-for-weblicht/tree/master/OpeNerServices` and `https://github.com/cnr-ilc/linguistic-tools-for-weblicht/tree/master/PanaceaServices`

However, in both examples, the `response` contains the analyzed text in the *tool native format*, which is KAF for OpeNer tools but can be any format for PANACEA ones. This is a PANACEA specific feature: actually, SoapLab simply wraps the original tools and produces the native output format, requiring only the implementation of converters to switch from one format to another.[18] Our implementation is

[18] For example, the Freeling service returns a tabbed file which must be converted to other formats such as TCF, KAF ...

not limited to simply wrap the offered Web Services;additional endpoints have been added to manage different input and output formats. This is required to fully integrate our tools in both Language Resource Switchboard and WebLicht. Since input and output formats can be plain texts, TCF and KAF documents, the following POST and GET[19] Web Services "consume" plain, TCF and KAF documents to produce TCF, TAB (tabbed) and KAF valid documents:

POST Plain texts openerservice/tokenizer/runservice, panaceaservice/freeling_it/runservice

POST TCF and KAF documents openerservice/tokenizer/**[tcf|kaf]**/runservice, panaceaservice/freeling_it/**[tcf|kaf]**/runservice

GET Plain texts openerservice/tokenizer/lrs, panaceaservice/freeling_it/lrs

GET TCF and KAF documents openerservice/tokenizer/**[tcf|kaf]**/lrs, panaceaservice/freeling_it/**[tcf|kaf]**/lrs

4 Conclusion and Future Work

In this paper we described an initiative carried out at ILC4CLARIN, which aims at integrating Web Services from PANACEA and OpeNer into the CLARIN infrastructure. The overall strategy is to wrap existing Web Services within REST APIs, so that both Language Resource Switchboard and WebLicht can exploit the new services. We have included only *Freeling* for Italian from the PANACEA set of Web Services and a basic tokenizer from OpeNer one. However, this work helped us to structure the various building blocks (see the github repository) so that other services can be easily wrapped, being them from PANACEA or OpeNer.

Truth be told, ILC4CLARIN already offers a tokenizer [20], which is a Java porting of the original OpeNer tokenizer. This service is in WebLicht [21] and was successfully tested for Language Resource Switchboard. So, what are the reasons that led us to use a different strategy? For PANACEA nothing different had to be done while, for OpeNer, the use of the java porting technique required that every original service was rewritten in Java, regardless of the original programming language. This was essentially the motivation that led us to decide to wrap the original services instead of rewriting them.

ILC4CLARIN uses dockers[22] for the services it offers and publishes the images in dockerhub[23]. We will follow this trend for the new services as well.

References

[Bosma et al.2009] Bosma W., Vossen P., Soroa A., Rigau G., Tesconi M., Marchetti A., Monachini M. and Aliprandi C. 2009. KAF: A Generic Semantic Annotation Format. In *Proceedings of the GL2009 Workshop on Semantic Annotation*.

[Ide and Suderman2007] Ide N. and Suderman K. 2007. GrAF: A Graph-based Format for Linguistic Annotations. In *Proceedings of the Linguistic Annotation Workshop*, LAW '07, pages 1-8. Stroudsburg, PA, USA. Association for Computational Linguistics.

[Hinrichs et al.2010] Hinrichs M., Zastrow T. and Hinrichs E. 2010. WebLicht: Web-based LRT Services in a Distributed eScience Infrastructure. In Nicoletta Calzolari (Conference Chair) at al., editors *Proceedings of the Seventh International Conference on Language Resources and Evaluation*. Valletta, Malta, May 19-21. European Language Resources Association (ELRA).

[Zinn2018] Zinn C. 2018. The Language Resource Switchboard. *Comput. Linguist.*, 44(4):631-639.

[Chandra et al.1981] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. 1981. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133.

[19] GET endpoints have been set up for eventual integration into Language Resource Switchboard (LRS)

[20] `https://ilc4clarin.ilc.cnr.it/services/ltfw-it/`

[21] `http://hdl.handle.net/20.500.11752/ILC-85@format=cmdi`

[22] We use rancher (version 1 (`https://rancher.com/`)) to manage docker images and composition

[23] `https://hub.docker.com/r/cnrilc/ltfw`