Distributing Text Mining tasks with *librAIry*

Carlos Badenes-Olmedo
cbadenes@fi.upm.es
Universidad Politécnica de Madrid
Ontology Engineering Group
Boadilla del Monte, Spain

José Luis Redondo-García
jlredondo@fi.upm.es
Universidad Politécnica de Madrid
Ontology Engineering Group
Boadilla del Monte, Spain

Oscar Corcho
ocorcho@fi.upm.es
Universidad Politécnica de Madrid
Ontology Engineering Group
Boadilla del Monte, Spain

ABSTRACT

We present *librAIry*, a novel architecture to store, process and analyze large collections of textual resources, integrating existing algorithms and tools into a common, distributed, high-performance workflow. Available text mining techniques can be incorporated as independent plug&play modules working in a collaborative manner into the framework. In the absence of a pre-defined flow, *librAIry* leverages on the aggregation of operations executed by different components in response to an emergent chain of events. Extensive use of Linked Data (LD) and Representational State Transfer (REST) principles are made to provide individually addressable resources from textual documents. We have described the architecture design and its implementation and tested its effectiveness in real-world scenarios such as collections of research papers, patents or ICT aids, with the objective of providing solutions for decision makers and experts in those domains. Major advantages of the framework and lessons-learned from these experiments are reported.

CCS CONCEPTS

•Applied computing → Document management and text processing; •Computer systems organization → *Architectures ;*

KEYWORDS

large-scale text analysis; NLP; scholarly data; text mining; data integration

1 INTRODUCTION

Given the huge amount of textual data about any domain that is daily being produced or captured in any imaginable domain, it becomes crucial to provide mechanisms for programmatically processing this raw data so we can make sense out of it: discarding all the noisy, non-relevant information and keeping only the data that can bring value for the involved agents (general consumers, experts, companies, investors...). While some specific tools already allow for advanced sense-making operations, others opt for composing a solution where different analysis techniques are integrated under a uniform data schema. However, this integration involves significant efforts on reconciling data sources, coordinating processing operations, and efficiently exploiting results from the execution of those techniques. There is the need for a more flexible paradigm where tools and algorithms for textual document analysis, from different programming languages and technologies, can operate independently and in a collaborative manner creating a common document oriented work-flow through their actions. In the context of the scientific publications, the personalized recommendation of research papers based on their content is a key novel feature for performing a smart selection of relevant resources over very big collections of scientific content. From the set of values and different attributes extracted from the papers and by generating advanced knowledge models about the information they contain we can bridge across the different relevant pieces of information and allow users to navigate them in a more efficient and powerful way. This knowledge about a specific document is frequently acquired by different techniques focused on revealing certain aspects of it, that are later combined to achieve one particular task. The architecture presented in this paper aims to ease the way different software modules work together and lays the foundation for efficiently process big volumes of textual documents in a distributed, decoupled manner.

2 RELATED WORK

The annotation of human-readable documents is a well-known problem in the Artificial Intelligence domain in general and Information Retrieval and Natural Language Processing fields in particular. There already exist a broad set of tools and frameworks able to analyze text for automatically producing such annotations, at very different levels of granularity: from minimal units such as terms and entities, to descriptors at the level of the entire collection such as topics or summaries. For example, StanfordNLP [7] framework allows to perform different operations such as PoS or Named Entity Recognition in various languages. Others like Mallet[1] or SparkLDA[2] perform topic modeling and clustering. The system we propose looks at the transversal problem of making those standalone tools coexisting under the same solution. Being able to effectively integrating them under a common ecosystem helps to seamlessly obtain different kind of annotations and boost the way those solutions can make sense of document collections.

Certain systems among the research and industrial communities have already integrated some of the annotation tools introduced

This work is supported by project Datos 4.0 with reference TIN2016-78011-C4-4-R, financed by the Spanish Ministry MINECO and co-financed by FEDER.
Author's addresses: C. Badenes-Olmedo and J.L. Redondo-García and O. Corcho , Ontology Engineering Group, Universidad Politécnica de Madrid.

DocEng '17, September 04–07, 2017, Valletta, Malta.

DOI: https://doi.org/10.1145/3103010.3121040

[1] http://mallet.cs.umass.edu
[2] https://spark.apache.org/mllib/

above. For example, [2] works with records from the biomedical domain, where robustness and high precision are prioritized. Therefore they rely on techniques supported by GATE[3] framework, which widely supports hand-crafted, domain specific techniques such as rules or finite state transducers. On the other side of the spectrum we find [6], where the authors try to annotate text from a much noisier, sparser and error-prone medium: a tweet stream. Therefore they do not rely on any linguistic feature, due to the unpredictable way short social media post are written. We observe how each of those examples has very specific needs and leverages on certain annotation tools in order to accomplish the tasks it was originally created for. In both systems the involved components are highly coupled so they can not be easily extended to contemplate complementary annotation tools or alternative modules. On the contrary, *librAIry* advocates loosely interconnected components that make the architecture more reusable and expandable in other systems across domains.

One crucial problem regarding the re-usability and expansion possibilities of those systems and the tools they leverage on is the language they have been developed in. For example, Mallet uses Java, but others like spaCy [4] are python-based. To the best of our knowledge, there has not been any significant efforts on reconciling into a single architecture such heterogeneous set of tools, therefore minimizing the engineering effort and maximizing scalability of the system so it can be applied to very different domains and textual annotation tasks.

In addition, available annotation systems rely on certain storage solutions that are suited for some tasks but are less adequate others. For example [5] uses a relational database (MySQL[5]) to ensure reliability and speed in managing the indexed information. In [8], the authors leverage on Virtuoso triple-store to provide native graph operations over the data. But new requirements may be considered for those systems so different storage needs can come into play. For example, column oriented databases (Cassandra[6]) can help to better handle high-volume queries on specific data fields. Same goes with text oriented indexes such as ElasticSearch [7], which can provide customized text-based search operations over the available information. *librAIry* straightforward supports the coexistence of different storage solutions, so it can be agnostic to the kind of underlying storage modules implemented. Thanks to the distributed nature of the proposed architecture, different databases can be synchronized under the same common environment working together to store and deliver results in a more efficient manner.

3 LIBRAIRY

librAIry is a framework where different text mining tools, available in various languages and technologies, can operate in a distributed, high-performance and isolated manner creating a common workflow through their actions. Instead to work towards a pre-defined sequence of actions, synchronization across modules is achieved through the aggregation of the operations executed by them in response to an emergent chain of events. This raises both technical and functional challenges to coordinate multiple executions. From the technical point of view, isolated environments and communication mechanisms are provided so initially dissimilar tools can be executed with maximum guarantees. From the functional point of view, all executions are coordinated to reach a final result as aggregation of partial results derived from each execution.

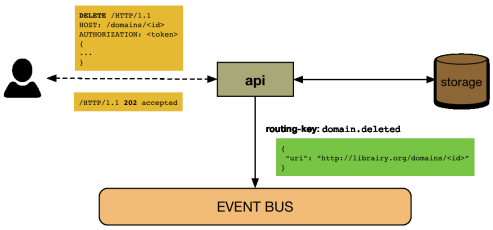

Figure 1: Domain deleted flow.

3.1 Functional Features

The architecture is articulated around three main concepts: (1) the **resource** such as *document*, a *part* of a *document*, or a *domain*. (2) the **actions** performed over them: *create*, *update* or *delete* a resource. And (3) the new **state** that is reached by the resource after an action is performed, such as *created*, *updated* or *deleted*. An **event** is a message containing details about those three aspects, published on a shared event-bus available for all the modules deployed in the framework. This will, in turn, allow that any module can perform actions on one or more resources in response to a new state reached by a given resource. Actions executed in parallel from distributed environments.

3.1.1 Resources. Two main kinds of resources are considered: those derived from external sources such as (1) *documents* from textual files (e.g. a research paper), (2) *parts* from logical divisions of a *document* (e.g. rhetorical classes or sections), and (3) *domains* from sets of *documents* (e.g. a conference or journal), and those derived from processing the previous ones such as *annotations*.

To better illustrate this model, consider to explore the research papers published at the SIGGRAPH conference in 2016. First, every paper will be materialized as a new *document* containing the full-text. Immediately after, the *document* will be automatically associated to several *parts*, each of them grouping sentences by rhetorical class (e.g. approach, background, challenge, future work and outcome) and by section (e.g abstract, introduction). Finally, a new *domain* will be created grouping all these *documents*. Different analysis will be performed extending the initial set of resources with more annotations at several representational levels: at *document level*, full-text based annotations are provided such as named-entities, compounds and descriptive tags. At *relational level*, connection between resources are found (e.g. semantic similarity-based relationships). And finally, at *domain level* annotations such as tags and summaries are composed describing the corpus of *documents*.

3.1.2 Event-based Paradigm. An event illustrates a performed action, i.e. a resource and its new state. It follows the Representational State Transfer (REST)[4] paradigm, but taking into account

[3]https://gate.ac.uk/
[4]https://spacy.io
[5]https://www.mysql.com/
[6]http://cassandra.apache.org
[7]https://www.elastic.co

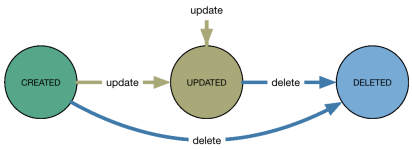

Figure 2: Resource states.

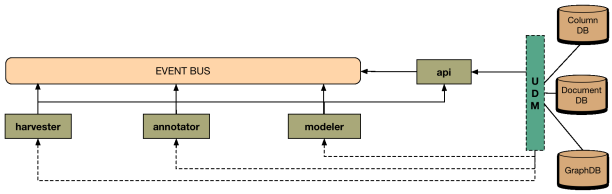

Figure 3: Modules.

the state reached after an action, i.e *created*, *deleted* or *updated*. Thus, an event contains the resource type and the new state reached by a specific resource.

3.1.3 Linked Data Principles . Data in *librAIry* is individually addressable and linkable [9] following the Linked Data principles defined by T. Berners-Lee [1]. Thus, resources (i.e. a *domain*, a *document*, a *part* or an *annotations*) have: (1) a URI as name, (2) a retrievable (or dereferenceable) HTTP URI so that it can be looked up, (3) a useful information provided by using standard notation (e.g. JavaScript Object Notation (JSON)) when it is looked up by URI, and (4) links to other URIs so that other resources can be discovered from it.

3.2 Framework Architecture

Following a publisher/subscriber approach, all the modules in the framework can publish and read events to notify and to be notified about the state of a resource. Therefore, the system flow is not unique and is not explicitly implemented, instead distributed and emergent flows can appear according to particular actions on resources.

3.2.1 Event-Bus. We use the Advanced Message Queuing Protocol (AMQP) as the messaging standard in *librAIry* to avoid any cross-platform problem and any dependency to the selected message broker. This protocol defines: *exchanges*, *queues*, *routing-keys* and *binding-keys* to communicate publishers and consumers.A message sent by a publisher to an exchange is tagged with a routing-key. Consumers matching that routing-key with the binding-key used to link the queue to that exchange will receive the message. In *librAIry* this key follows the structure: *resource.status*.Since a wildcard-based definition can be used to set the key, this paradigm allow modules both listening to individual type events (e.g. ́domains.created́for new domains), or multiple type events (e.g. #.created for all new resources).

3.2.2 API. A HTTP-Rest Application Program Interface (API) was designed for interaction with end-users. Any external operation motivated by a user will be handled here. Some of them, usually those related to reading operations, will be completely managed by this module getting all the data from the internal storage. However, those operations implying a modification of the status of some resource (e.g. creation of a *document*), may be also performed by other modules listening for that type of event asynchronously. This module publishes to the following routing-keys: *domain.(created;updated;deleted)*, *document.(created;updated;deleted)*, *part.(created;updated;deleted)*, and *annotation.(created;updated;deleted)*.

3.2.3 Storage. Multiple types of data can be handled in this ecosystem. Inspired in the Data Access Object (DAO) pattern, we have created a Unified Data Manager (UDM) providing access to any type of data used in the system. Three types of databases have been considered:

- **column-oriented database**: Focused on unique identified and/or *structured data*. This storage allow us searching key elements across resources.
- **document-oriented database**: Focused on indexing raw text. This storage allow us to execute advanced search operations over all the information gathered about a textual resource.
- **graph database**: Focused on relations. This storage allow us exploring resources through the relationships between them.

3.2.4 Modules. The modules composing *librAIry* have been designed following the microservices architectural style. A module is a cohesive (i.e. it implements only functionalities strongly related to the concern that it is meant to model [3]) and independent process working on the framework with a specific purpose. This purpose is defined by both the routing-key and the binding-key associated to the events handled by the module.

These are the main types of modules identified in *librAIry*:

- **Harvester**: creates system resources such as *documents*, *parts* and *domains*, from local or remote located textual files.
  - Listening for: nothing
  - Publishing to: *document.(created)*, *part.(created)*, *domain.(created;updated)*
- **Annotator**: retrieves named-entities, compounds, lemmas and other annotations resulting of Natural Language Processing (NLP) task execution from *documents* and *parts*.
  - Listening for: *document.(created;updated)*, *part.(created;updated)*
  - Publishing to: *annotation.(created;deleted)*
- **Modeler**: builds representational models from a given *domain*.
  - Listening for: *domain.(created;updated)*
  - Publishing to: *annotation.(created;deleted)*

4 EXPERIMENTS AND LESSONS-LEARNED

librAIry has been used in some real scenarios such as a research-paper repository for the European project DrInventor [8], a support to decision makers for analyzing patents and public aids for the

[8] http://drinventor.eu

ICT sector, and also as a book recommender for an online content platform. This has allowed us to identify some weak and strong points of the framework and iterate over the architecture to come with the described solution.

The following modules have been developed[9]: (1) a ***general-purpose harvester*** which retrieves text and meta-information from PDF files in local or remote file-system; (2) a ***research paper-oriented harvester*** focused on collecting and processing more specific textual files (e.g. scientific papers) creating both *documents* and *parts* inferred from the rhetorical classes of the paper; (3) a ***Stanford CoreNLP-based Annotator*** which discovers named-entities, compounds and lemmas from *documents* and *parts*; (4) a ***Topic Modeler*** based on Latent Dirichlet Allocation (LDA) which creates probabilistic topic models for each *domain* in the framework. They are annotated with the set of topics (i.e. ranked list of words) discovered from the corpus, and both *documents* and *parts* of that domain are also annotated by the vector of probabilities to belong to these topics. It uses the Spark implementation of the algorithm; and (5) a ***Word Embedding Modeler*** which creates a *word2vec* model from the *documents* contained in a *domain*.

Due to linear scalability and high performance features, Cassandra has been used to support the column-oriented storage functionality, Elasticsearch as document-oriented storage and Neo4j as graph-oriented storage.

All modules in *librAIry* have been packaged as Docker [10] containers and uploaded to Docker-Hub [11] to facilitate the installation of the system.

Maximizing information re-usability and minimize irrelevant data, becomes specially important when the system handles large collections of data (around million of documents). Fine-grained resource definitions have been key to achieve this, so modules execute actions only when really necessary. When a new *domain* is created, for instance, a new Topic Model is trained for that *domain* and is used to calculate the semantic similarity between the *documents* (and the *parts*) in that domain. If a new *document* (or *part*) is added to that *domain*, the model is trained again and the semantic similarities are re-calculated. However, this becomes unfeasible when the domain is frequently updated and it is composed by a large number of documents. One solution has been to define a new type of resource between domains and documents, models, that describes the representational state (e.g. topic model) of a collection of documents. Thus the model is only re-trained when a significant amount of *documents* are added to the sampling data set and not to the entire *domain*. This less transient model is used to calculate semantic similarities between the *document* collection (and *parts*) inside a *domain* in a more efficient way. Following this more precise execution of tasks, the routing-keys should include the URI of the implied resource into the definition, not only in the content of the message. It would allow modules listening to both the type of a resource or to a specific resource (or subsets, via regular expressions).

While the storage modules are always used to save/update/delete a resource, they are not always required from the end-user. The graph storage, for instance, makes sense when a path between two *documents* or *parts* is requested for a given *domain*. However, some *domains* are not intended to be explored by their linked resources. A more fine/grained definition of resources will allow graph-storage being only used when necessary.

On the other hand, distributed execution of NLP tasks (not only in threads, but also in machines) has proved to be especially useful to handle large collection of *documents*. It requires less processing time than a monolithic solution (e.g. CoreNLP application) and it also provides a dynamic load balancing between modules.

5 CONCLUSIONS AND FUTURE WORK

In *librAIry*, existing algorithms and tools coming from different technologies can work collaboratively to process and analyze large collections of textual resources which has been successful applied to some real scenarios [12].

A new model definition based on the previously mentioned principle of maximizing information re-usability and minimize irrelevant data is being studied to create a more fine-grained resource design. New domains, in the sense of particular vocabularies or specific textual formats, are also being analyzed to be included into the system via specific harvesters and/or more precise annotators. Moreover, a template-based mechanism oriented to facilitate the integration of new tools and techniques into the system is being built to make easier to develop new modules as well as increasing the available modules at Docker-Hub.

REFERENCES

[1] Christian Bizer, T Heath, and T Berners-Lee. 2009. Linked data-the story so far. *International journal on Semantic Web and Information Systems* 5, 3 (2009), 1–22. DOI: http://dx.doi.org/10.4018/jswis.2009081901

[2] Hamish Cunningham, Valentin Tablan, Angus Roberts, and Kalina Bontcheva. 2013. Getting More Out of Biomedical Documents with GATE's Full Lifecycle Open Source Text Analytics. *PLOS Computational Biology* (2013). DOI: http://dx.doi.org/10.1371/journal.pcbi.1002854

[3] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. 2016. Microservices: yesterday, today, and tomorrow. *CoRR* abs/1606.0 (2016), 1–17. DOI: http://dx.doi.org/10.13140/RG.2.1.3257.4961

[4] Roy T Fielding and Richard N Taylor. 2002. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology* 2, 2 (2002), 407–416. DOI: http://dx.doi.org/10.1145/514183.514185

[5] Laura I Furlong, Holger Dach, Martin Hofmann-Apitius, and Ferran Sanz. 2008. OSIRISv1. 2: a named entity recognition system for sequence variants of genes in biomedical literature. *BMC bioinformatics* 9, 1 (2008), 84.

[6] Chenliang Li, Jianshu Weng, Qi He, Yuxia Yao, Anwitaman Datta, Aixin Sun, and Bu-Sung Lee. 2012. TwiNER: Named Entity Recognition in Targeted Twitter Stream. ACM, New York, NY, USA, 721–730. DOI: http://dx.doi.org/10.1145/2348283.2348380

[7] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. DOI: http://dx.doi.org/10.3115/v1/P14-5010

[8] Giuseppe Rizzo, Raphäel Troncy, Oscar Corcho, Anthony Jameson, Julien Plu, Juan Carlos Ballesteros Hermida, Ahmad Assaf, Catalin Barbu, Adrian Spirescu, Kai-Dominik Kuhn, and others. 2015. 3cixty@ Expo Milano 2015: Enabling Visitors to Explore a Smart City. (2015).

[9] S Turchi, L Ciofi, F Paganelli, F Pirri, and D Giuli. 2012. Designing EPCIS through Linked Data and REST principles. *Software, Telecommunications and Computer Networks ({SoftCOM}), 2012 20th International Conference on* (2012), 1–6.

[9] https://github.com/librairy

[10] https://www.docker.com

[11] https://hub.docker.com/u/librairy/

[12] http://drinventor.dia.fi.upm.es