Promoting Scientific Creativity by Utilising Web-based Research Objects

Project acronym: Dr Inventor

Deliverable No. 5.4
Repository of indexed ROs

Grant agreement no: 611383

Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	Dr Inventor
Project Full Name:	Promoting Scientific Creativity by Utilising Web-based Research Objects
Deliverable No.:	D5.4
Document name:	Repository of indexed ROs
Nature (R, P, D, O)[1]	P
Dissemination Level (PU, PP, RE, CO)[2]	PU
Version:	1
Actual Submission Date:	23/12/2015
Editor: Institution: E-Mail:	Carlos Badenes Ontology Engineering Group (OEG), Universidad Politécnica de Madrid (UPM), Madrid, Spain cbadenes@fi.upm.es

ABSTRACT:
This deliverable describes the design and implementation of the unified information service whose objective is to simplify the search of relevant content in situations where multiple heterogeneous non-structured information sources are involved, such as web-based Research Objects.

KEYWORD LIST:
Semantic publishing, Research Object repository

[1] **R**=Report, **P**=Prototype, **D**=Demonstrator, **O**=Other

[2] **PU**=Public, **PP**=Restricted to other programme participants (including the Commission Services), **RE**=Restricted to a group specified by the consortium (including the Commission Services), **CO**=Confidential, only for members of the consortium (including the Commission Services)

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7-ICT-2013.8.1) under grant agreement n° 611383.

MODIFICATION CONTROL			
Version	**Date**	**Status**	**Author**
0.1	10/12/2015	Draft	Carlos Badenes
0.2	16/12/2015	Draft	Rafael Gonzalez
0.3	23/12/2015	Draft	Carlos Badenes
1.0	29/12/2015	Final	Carlos Badenes
1.1	31/12/2015	Reviewed	Carlos Badenes
1.2	28/02/2017	Extended	Carlos Badenes

List of contributors

- Oscar Corcho, OEG, UPM.
- Rafael Gonzalez, OEG, UPM.
- Carlos Badenes, OEG, UPM.
- Feng Dong, University of Bedfordshire.

Contents

1 Executive Summary

This deliverable describes the unified information access middleware whose objective is to simplify the search of relevant content in situations where multiple heterogeneous non-structured information sources are involved, such as web-based Research Objects (ROs) .

It combines access and search activities to different information sources, that are associated to the products of scientific research, encapsulating and relating the meaning of the knowledge objects that they contain, offering thus:

- A structured information space view, where the involved entities, their taxonomies and topics are explicitly annotated and can be used to explore the information space.
- A coherent information space view that abstracts away from the multiple information sources, providing a single search and access interface in a transversal content-oriented manner.

The service is available via a REST API at http://drinventor.dia.fi.upm.es/api , the code is available at https://github.com/librairy , and some results are showed at http://drinventor.dia.fi.upm.es .

2 Structure of the deliverable

This document is organized as follows: Section 3 shows technical details about the proposed system. In Section 4 we describe the Application Programming Interface (API) used to explore documents and Section 5 presents a complete use-case from the point of view of an end-user. Finally, Section 6 explains the steps to install and run the service.

3 Design

The *Repository of Indexed Research Objects* (**librairy**) is a system composed by six loosely-coupled modules connected by an event-bus using standardized data protocols and formats.

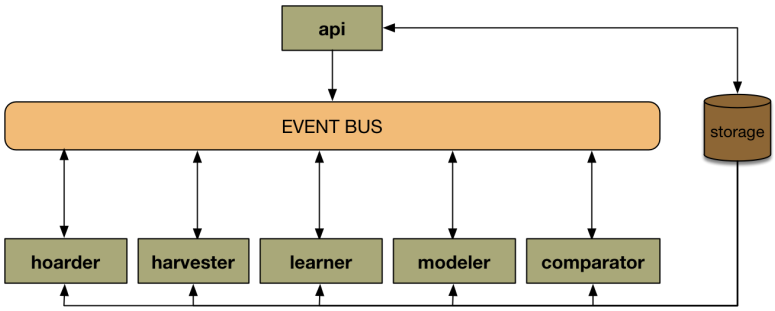

Figure 1. System Architecture

It follows a ***Staged Event-Driven Architecture*** (SEDA) [1] that decomposes the flow into a set of modules connected by queues. When a new research content is published or modified from a *source* or when someone submits a new *document*, a new event is created and handled in the system to update or to build new relationships between this new resource and others.

Next we provide a list of modules and their responsibility:

- **Api**: deploys the web interface to allow executing operations on the system.
- **Hoarder**: pools external repositories to download resources.
- **Harvester**: retrieves text and meta-information from resources to create domain entities.
- **Learner**: identifies relevant terms and discovers relations among them from a domain.
- **Modeler**: creates internal models to represent and categorize domain entities.
- **Comparator**: measures the analogy between domain entities according to the models.

3.1 Resources

The system is updated with both gathered and submitted information internally modeled as *resources*. A resource is an abstract representation of the information described by the type of content and the status.

The states are common for all, but the type of content depends on each of them:

- ***Document***: meta-information retrieved from a research object. A *document* is composed by a set of *items*.
- ***Item***: each of the elements that make up a research object (i.e. a *document*) such as a paper, programming-code, an image, a workflow, and so on.

- ***Part***: logical division inside an *item* based on aspects such as sections of the research article (i.e. *abstract , introduction, method, results, discussion* and *conclusion*) or the rhetorical class of the sentences[3] (i.e. *approach, background, challenge, future-work* and *outcome*).
- ***Word***: a *term*, *entity* or any other textual unit that composes a text.

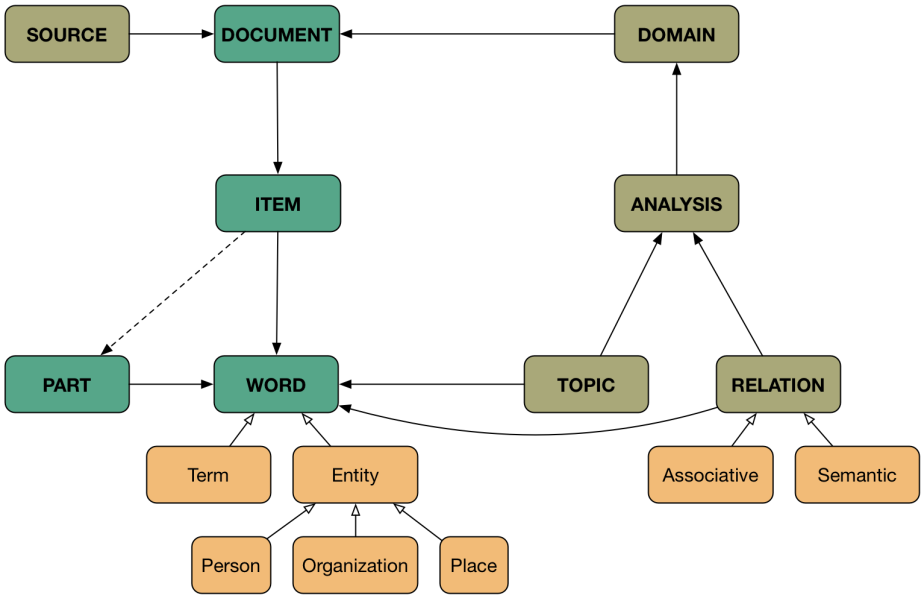

Figure 2. System Resources

- ***Source***: repository of *documents*.
- ***Domain***: set of *documents*.
- **Topic:** abstract concept described by a set of *words* that represents a research area or subject in a *domain*.
- **Relation:** *associative* or *semantic* connection between *words* in a *domain*.
- **Analysis**: study performed in a *domain*. Responsible for the creation of *topics* and *relations*.

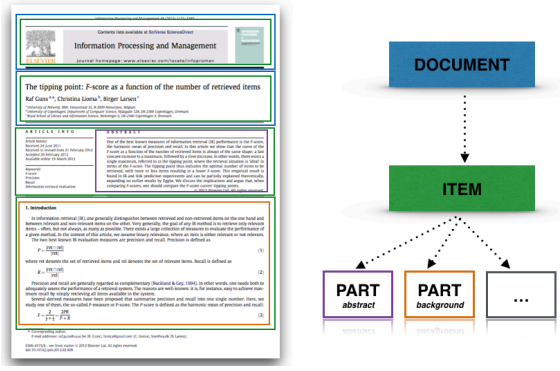

Figure 3. Representation of a paper in the system.

[3] http://backingdata.org/dri/library/1.0.5/doc/edu/upf/taln/dri/lib/model/ext/RhetoricalClassENUM.html

To better understand this representation, let's see an example. Consider that we want to add a paper (*like the pdf file seen in the Figure 3*) to the system. A new *document* will be created with information about the title, the author(s), the publisher and the language.

A new *item* containing, in this case, the same title, the same authors, the published-date, the language, the format (*pdf*) and the key-words will be also created. Moreover, this *item* will be associated to several *parts,* each of them grouping sentences by their rhetorical class (e.g. *approach, background, challenge, future work* and *outcome*) or by their section (e.g *abstract, introduction*).

3.1.1 Source

A *source* is a repository of research objects. It contains the following information:

- **uri**: the *Uniform Resource Identifier* created by the system to uniquely identify it. It must be a *Universally Unique Identifer* (UUID) along with a prefix identifying the type of the resource: e.g `sources/de305d54-75b4-431b-adb2-eb6b9e546014`
- **creation-time**: date on which this resource was created. It must be a formatted timestamp following ISO-8601[4].
- **name**: a label associated to the resource.
- **description**: additional information about it.
- **url**: the *Uniform Resource Locator* of the repository.
- **protocol**: defines how the digital content is published.

A *source* may contain zero or more references to *documents* and a *document* may have one or more references to *sources,* i.e. the same *document* can be available in more than one *source*.

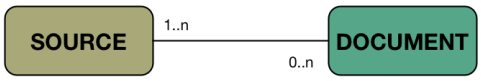

Figure 4. Range and domain of a *Source*.

It may be a *static* or a *dynamic* repository:

- ***Static***: repository that will not change along time. So, once processed, new information will never be available from it again. It can be a single file (e.g. `http://world.std.com/~rjs/indinf56.pdf`) or a closed time-based expression for an open archive service (e.g. `http://www.worldsciencepublisher.org/journals/index.php/AASS/oai?from=2012-01-01T00:00:00`).
- ***Dynamic***: repository that may have new documents in the future, such as an open archive publisher (e.g. `http://oa.upm.es/perl/oai2`), a RSS feeder (e.g. `http://rss.slashdot.org/Slashdot/slashdot`) , a remote folder (e.g. `//192.168.5.125/Public/papers`) or even a web page (e.g. `https://en.wikipedia.org/wiki/Artificial_Intelligence`). This type of resources will be continuously *polled* by the *hoarder* module.

[4] http://www.iso.org/iso/home/standards/iso8601.htm

Currently, *The Open Archives Initiative Protocol for Metadata Harvesting* (OAI-PMH)[5] and *Really Simple Sindication* (RSS)[6] are the protocols supported by our system. Future integrations will be done to allow sources such as *Dropbox*[7], *CIFS/SMB*, *FTP/FTPS*, *Elsevier API*[8], *Figshare API*[9], *arXiv API*[10], *DBLP Corpora*[11], *Research Gate*, *Mendeley*[12] and *CiteSeer*[2]. More details about how to include a new protocol are shown in the *Hoarder Section* of this document.

3.1.2 Domain

A *domain* is a logical grouping of *documents*. It defines the workspace for the *modeler* and the *learner* module. By default, all *sources* have an associated *domain* with their *documents*.

It contains the following information:

- **uri**: the *Uniform Resource Identifier* created by the system to uniquely identify it. It must be a *Universally Unique Identifer* (UUID) along with a prefix identifying the type of the resource: e.g `domains/de305d54-75b4-431b-adb2-eb6b9e546014`
- **creation-time**: date on which this resource was created. It must be a formatted timestamp following ISO-8601.
- **name**: a label associated to the resource.
- **description**: additional information about it.

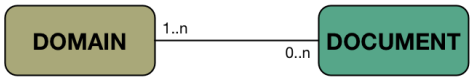

Figure 5. Range and domain of a *Domain*.

Furthermore, a *domain* can contain zero or more references to *documents* and a *document* may be referenced by one or more *domains.*

3.1.3 Document

A *document* contains all the meta-information associated to the publication, according to the *Open Archives Initiative for Object Reuse and Exchange*[13] (OAI-ORE) manifest for research objects and the *Dublin-Core Metadata Annotation*[14], it could include:

[5] http://www.openarchives.org
[6] http://www.rssboard.org/rss-specification
[7] https://www.dropbox.com/developers
[8] http://dev.elsevier.com/
[9] http://api.figshare.com/docs/intro.html
[10] http://arxiv.org/help/api/index
[11] http://dblp.uni-trier.de/faq/How+can+I+download+the+whole+dblp+dataset
[12] http://dev.mendeley.com
[13] http://www.openarchives.org/ore/1.0/toc.html
[14] http://dublincore.org/documents/1999/07/02/dces/

- **uri**: the *Uniform Resource Identifier* created by the system to uniquely identify it. It must be a *Universally Unique Identifer* (UUID) along with a prefix identifying the type of the resource: e.g `documents/de305d54-75b4-431b-adb2-eb6b9e546014`
- **creation-time**: date on which this resource was created. It must be a formatted timestamp following ISO-8601.
- **publishedOn**: the time the resource was published. t must be a formatted timestamp following ISO-8601.
- **publishedBy**: an entity responsible for making the *document* available. It can be a person, an organization or a service. It may be different from the entity that conceptually formed the resource (e.g. wrote the document), which should be recorded as *authoredBy*. This entity should be identified by a valid *Uniform Resource Identificator* (URI), e.g. *WebId*[15], orcid[16] or internal URI.
- **authoredOn**: the time the research was conceptually formed. The author time should be present if different from *publishedOn*. It must be a formatted timestamp following ISO-8601.
- **authoredBy**: an entity primarily responsible for making the content of the document. It may be a list to indicate multiple authors. Each of them identified by a valid URI, e.g. *WebId*[17] , orcid[18] or internal URI.
- **retrievedFrom**: a URI identifying the *source* from which the *document* was derived. This property should be accompanied with *retrievedOn*.
- **retrievedOn**: the time the *document* was retrieved on. If this property is present, then *retrievedFrom* must also be present. It must be a formatted timestamp following ISO-8601.
- **format**: the physical or digital manifestation of the resource. Typically, it includes the media-type, i.e. the IANA[19] code, of the *document*.
- **language**: the language(s) in which the document is specified. It is defined by RFC-1766[20] which includes a two-letter language code followed, optionally, by a two-letter country code.
- **title**: a name given to the *document*. It is a name by which the *document* is formally known.
- **subject**: keywords, key phrases or classification codes annotated by the authors that describe a topic of the resource.
- **description**: an account of the content of the *document*. It may include but is not limited to an abstract, or a free-text account of the content.
- **rights**: information about rights held in and over the *document*.

Furthermore , a *document* may contain zero or more *items.* In turn, an *item* can belong to one or more *documents*.

[15] http://www.w3.org/wiki/WebID
[16] http://orcid.org
[17] http://www.w3.org/wiki/WebID
[18] http://orcid.org
[19] http://www.iana.org/assignments/media-types/media-types.xhtml
[20] http://www.ietf.org/rfc/rfc1766.txt

Since ***librairy*** can also discover analogies among *documents*, a *document* may contain zero or more references to other *document*s.

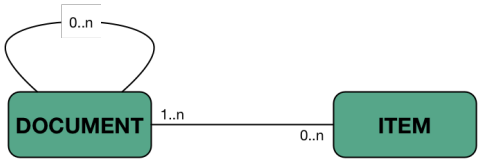

Figure 6. Range and domain of a *document*.

3.1.4 Item

An *item* is each of the elements that make up a *document* (i.e. the files bundled in a research object). It may be a paper, programming-code, an image, a workflow or any other media content.

It contains the following information:

- **uri**: the *Uniform Resource Identifier* created by the system to uniquely identify it. It must be a *Universally Unique Identifer* (UUID) along with a prefix identifying the type of the resource: e.g `items/de305d54-75b4-431b-adb2-eb6b9e546014`
- **creation-time**: date on which this resource was created. It must be a formatted timestamp following ISO-8601.
- **authoredOn**: the time the research was conceptually formed. The author time should be present if different from *publishedOn*. It must be a formatted timestamp following ISO-8601.
- **authoredBy**: an entity primarily responsible for making the content of the document. It may be a list to indicate multiple authors. Each of them identified by a valid URI, e.g. *WebId*[21] , orcid[22] or internal URI.
- **format**: the physical or digital manifestation of the resource. It includes the media-type, i.e. the IANA[23] code.
- **language**: the language(s) in which it is specified. It is defined by RFC-1766[24] which includes a two-letter Language code followed, optionally, by a two-letter country code.
- **title**: a name given to the *item*. It is a name by which the *item* is formally known.
- **subject**: keywords, key phrases or classification codes annotated by the authors that describe a topic of the resource.
- **description**: an account of the content of the *document*. It may include but is not limited to an abstract, or a free-text account of the content.
- **url**: path to the file. e.g. pdf file path, png file path.
- **content**: textual annotation about the *file*. When it is a paper, it contains the raw-text of the paper. When it is an image, it contains the textual description of the image.

[21] http://www.w3.org/wiki/WebID
[22] http://orcid.org
[23] http://www.iana.org/assignments/media-types/media-types.xhtml
[24] http://www.ietf.org/rfc/rfc1766.txt

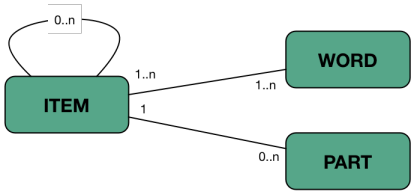

Figure 7. Range and domain of a *item*.

Furthermore, an *item* may contain zero or more *parts* and one or more *words.* In turn, a *part* only belongs to one *item,* and a *word* can belong to one or more *items*.

Since ***librairy*** can also discover analogies among *items*, an *item* may contain zero or more references to other *items*.

3.1.5 Part

A *part* is a logical section in an *item*. When an *item* is a paper, for instance, it will have as *parts* the rhetorical classes identified in the sentences of its textual-content. It contains the following information:

- **uri**: the *Uniform Resource Identifier* created by the system to uniquely identify it. It must be a *Universally Unique Identifer* (UUID) along with a prefix identifying the type of the resource: e.g `parts/de305d54-75b4-431b-adb2-eb6b9e546014`
- **creation-time**: date on which this resource was created. It must be a formatted timestamp following ISO-8601.
- **sense**: content-type. It could be a rhetorical class such as *background*, *approach*, *challenge*, *future-work* or *outcome*; or a section in the text such as *introduction*, *abstract, discussion, conclusion, results or method;* or any other label used to classify parts of a text.
- **content**: text retrieved from the text of the *item,* sharing the same class in a classification (i.e. content-type).

Furthermore, a *part* can contain one or more *words* and a *word* can be referenced by one or more *parts*.

Figure 8. Range and domain of a *part*.

Since ***librairy*** can also discover analogies among *parts*, a *part* may contain zero or more references to other *part*s.

3.1.6 Word

A *word* is a *term* or an *entity* (i.e. person, organization or place) or any other meaningful unit contained in a text. It may include linguistic annotations such as lemma, stem and part-of-speech (POS).

It contains the following information:

- **uri**: the *Uniform Resource Identifier* created by the system to uniquely identify it. It must be a *Universally Unique Identifer* (UUID) along with a prefix identifying the type of the resource: e.g `words/de305d54-75b4-431b-adb2-eb6b9e546014`
- **creation-time**: date on which this resource was created. It must be a formatted timestamp following ISO-8601.
- **content**: the alphanumeric character string.
- **lemma**: the word which stands at the head of a definition in a dictionary.
- **stem**: the root of the word.
- **pos:** (*part-of-speech*) the syntactic category of the word.
- **type**: *term* or e*ntity*.

3.1.7 Topic

A *topic* is an abstract concept described by a sorted list of *words* that represents a research area or subject in a *domain*. The order means the relevance of the *word* in the *topic.*

It contains the following information:

- **uri**: the *Uniform Resource Identifier* created by the system to uniquely identify it. It must be a *Universally Unique Identifer* (UUID) along with a prefix identifying the type of the resource: e.g `topics/de305d54-75b4-431b-adb2-eb6b9e546014`. URIs from external services describing synsets (e.g. BabelNet) will also be used in future .
- **creation-time**: date on which this resource was created. It must be a formatted timestamp following ISO-8601.
- **content**: an account of the meaning of the topic. Usually, it is the top 15 relevant words.

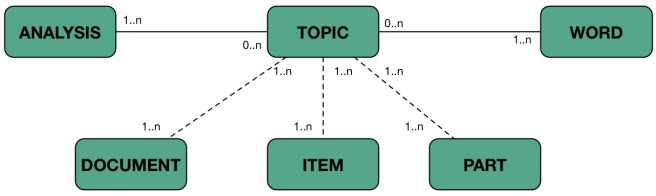

Figure 9. Range and domain of a *topic*.

Furthermore, a *topic* contains one or more *words* and one or more *analyses*. In turn, a *word* can be referenced by zero more *topics* and an *analysis* can be referenced by zero or more *topics*.

Since *topics* are also used to represent *resources,* they are also referenced by one or more *document*, *item* and *parts*.

3.1.8 Relation

A *relation* is an *associative* or *semantic* link between *words*. When they are entities, the *relation* will be *associative*. When they are terms, the *relation* will be *semantic*.

It contains the following information:

- **uri**: the *Uniform Resource Identifier* created by the system to uniquely identify it. It must be a *Universally Unique Identifer* (UUID) along with a prefix identifying the type of the resource: e.g `topics/de305d54-75b4-431b-adb2-eb6b9e546014`
- **creation-time**: date on which this resource was created. It must be a formatted timestamp following ISO-8601.
- **type:** associative or semantic.
- **describes**: more details about the relationship, e.g. antonymy, meronymy, etc.

Figure 10. Range and domain of a *relation*.

Furthermore, a *relation* contains two *words* and one or more *analyses*. In turn, a *word* can be referenced by zero or more *relations* and an *analysis* can be referenced by zero or more *relations*.

3.1.9 Analysis

An *analysis* is a study about the documents contained in a *domain.* It is mainly focused on *items* but may also include *parts* for deeper analyses . Its main purpose is to discover *topics* and *relations* in the *domain* and calculate the similarity values among its *documents*.

It may contain the following information:

- **uri**: the *Uniform Resource Identifier* created by the system to uniquely identify it. It must be a *Universally Unique Identifer* (UUID) along with a prefix identifying the type of the resource: e.g `analysis/de305d54-75b4-431b-adb2-eb6b9e546014`
- **creation-time**: date on which this resource was created. It must be a formatted timestamp following ISO-8601.
- **description**: details about the algorithm used for the analysis.
- **configuration**: details about the parameters of the algorithm.
- **report**: scheme containing a representation of each resource analyzed.

NOTE: Future works will include a more detailed parameterization of the algorithms performed during the analysis process.

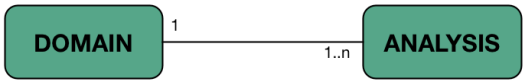

Figure 11. Range and domain of *analysis*.

Furthermore, an *analysis* contains only one *domain*, and can be referenced by one or more *domains*.

3.2 Status

A resource can have, at a time, one of these status:

- ***created***: initial action. The first status of a resource.
- ***updated***: the resource has changed.
- ***deleted***: the resource was removed.

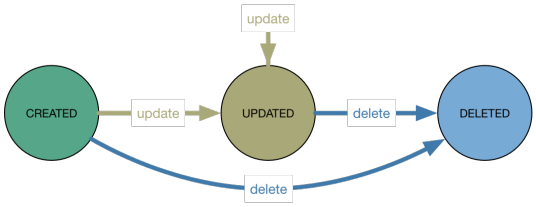

Figure 12. Resource status.

3.3 Event-Bus

Following a publisher/subscriber approach, all the modules in the system can publish and read events to notify and to be notified about the system state.

For example, when a new *domain* is created, an event-message will be published to the channel: `domain.created`. This event will be managed by the *learner* module to begin to analyze all the *items* in that *domain* and also by the *modeler* module to begin to build models which represent these *items*. Therefore, the system flow is not unique and is not directly implemented, instead parallel and emergent flows can appear according to particular actions on resources.

We use the ***Advanced Message Queuing Protocol***[25] (AMQP) as our messaging standard to avoid any cross-platform problem and any dependency to the selected message broker. This protocol defines the following elements: *exchange*, *queue* and *routing-key*.

In this scenario, a *producer* module sends a message to the *exchange* element, instead of a *queue*, along with a *routing-key*. That *exchange* may be bound to *queues* through *topic* and *group bindings* defined by a module. A *topic binding* is a directive indicating what messages should be routed from

[25] http://www.amqp.org

an *exchange* to a *queue*. The *group binding* allows system to deliver a message only to one consumer sharing the same *group* value, getting a dynamic distribution of load between them.

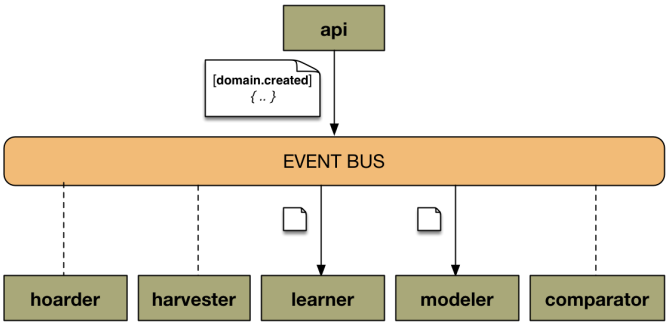

Figure 13. flow of the 'domain.created' event.

Consumer modules listening to *queues* bounded to the *exchange* by a *topic binding* that match to the *routing-key* used to sent the message, will receive that message.

We adopted the *exchange* paradigm for building our *event-bus.* Thus, every module (i.e. api, learner, modeler, etc) defines the kind of events to listen by the *binding-key* used to bound its *queue* to the *exchange*. The messages are sent with a routing-key to *queues* using a *binding-key* that matches with that *routing-key*. These *queues* are not shared among different modules, but yes among the instances of the same module when the system is running on a cluster.

Considering only systems that offer persistence and replication as well as routing, delaying and re-delivering implementing the AMQP protocol, we used the RabbitMQ Message Broker[26].

3.3.1 Routing-Key

In general, a *routing-key* can be defined by a list of words delimited by dots. In our case, it consists of only two words separated by one dot:

<resource>. <status>

The first part identifies the resource associated to the event, and the second one is its current status. For instace, `document.created`, `domain.updated`, `item.deleted`.

3.3.2 Binding-Key

An *exchange* is bound to *queues* through *binding-keys* composed by a ***topic-key*** and a ***group-key***. Messages will be delivered to those *queues* bound to the *exchange* with a binding-key that *matches* with the routing-key of the message.

The **topic-key** is defined in the same way that the routing-key, so it is also composed by two words separated by one dot:

[26] http://www.rabbitmq.com

<resource>.<status>

There are two special cases for matching a *topic-key*:

- '*****' : it means that it can be substituted by exactly one word (e.g. `*.created` listen for all new resources added to the system).
- '**#**' : it means that it can be substituted by zero or more words (e.g. '#' listen for all resources in any status).

The **group-key** is a text string that allows different clients to join in the same queue to distribute the events among them.

Complete examples about routing are shown in the next section "Scenarios".

3.3.3 Messages

Usually, the content of a message is a string of characters in a *Javascript Object Notation*[27] (JSON) format. But also, it may be an array of bytes serialized in a particular way. Our event-bus allows clients to use both formats. The type of the event, i.e. its *routing-key*, defines implicity the format of the message to be understood by both producer and consumer.

Except special cases, the message will be a text in JSON format with only one field: the URI of the resource.

```
{ "uri": "string" }
```

Thus, for example, the message associated to the creation of a new *source*, i.e *routing-key* equals to `source.created`, will be:

```
{ "uri": "sources/2233-23233-1192-30" }
```

3.3.4 Scenarios

In order to understand the behaviour of our event-bus, let's see some examples for the following configuration shown in figure 14.

3.3.4.1 S1: Different topic-keys and group-keys

In this case, the values may be:

- Topic_A = `document.created`
- Topic_B = `document.deleted`
- Group_A = `harvester`
- Group_B = `modeler`

[27] http://www.json.org

Then, when a producer sends a message to the routing-key: '`document.created`, only the *Consumer1* will receive the message. If the message had been sent to '`document.deleted`, only the *Consumer2* would have received the message.

So, in this scenario, consumers are listening for different messages.

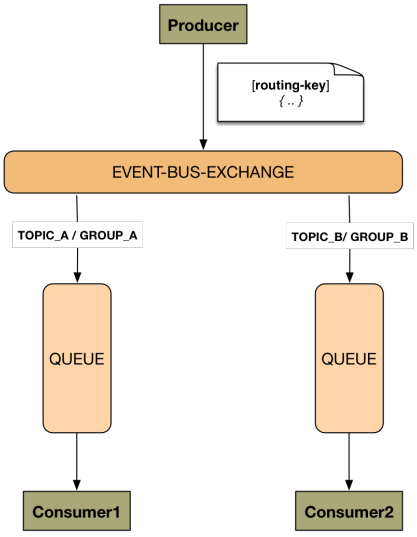

Figure 14. Producer and consumers connected to event-bus.

3.3.4.2 S2: Same topic-keys and different group-keys

In this case, the values may be:

- Topic_A = `document.created`
- Topic_B = `document.created`
- Group_A = `harvester`
- Group_B = `modeler`

When a producer sends a message to the routing-key: '`document.created`, both *Consumer1* and *Consumer2* will receive that message.

So, in this scenario, the message will be duplicated among consumers which have the same *topic-key*.

3.3.4.3 S3: Different topic-keys and same group-keys

In this case, the values may be:

- Topic_A = `document.created`
- Topic_B = `document.deleted`
- Group_A = `harvester`

- Group_B = `harvester`

When a producer sends a message to the routing-key: '`document.created`, only the *Consumer1* will receive the message. When a producer sends the message to the routing-key: '`document.deleted`, only the *Consumer2* will receive it.

In this scenario, the consumers are listening for different messages. The common *group-key* has no effect because the *topic-keys* are different.

3.3.4.4 S4: Same topic-keys and group-keys

In this case, the values may be:

- Topic_A = `document.created`
- Topic_B = `document.created`
- Group_A = `harvester`
- Group_B = `harvester`

When a producer sends a message to the routing-key: '`document.created`, only one of the consumers will receive the message. By default, a *round-robin* approach[3] is defined to dispatch this type of messages.

In this scenario, the message is balanced among consumers listening for the same route.

3.4 Modules

Currently, the service is composed by six modules, each of them with different responsibilities.

3.4.1 Api

This module provides an interface to the end-user. Any operation requested by the user will be handled by it. Some of these operations, usually those related to read some information about a resource, will be directly managed by this module getting the data directly from the internal storage.

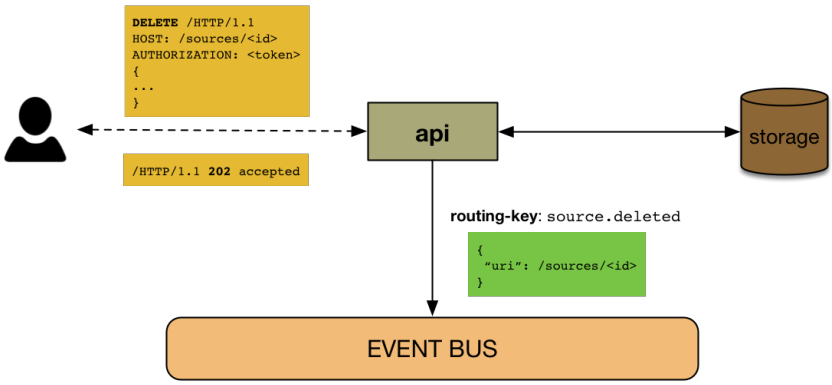

Figure 15. Workflow associated to a DELETE operation from the API module.

However, those operations that imply a modification of the status of some resource, will be performed by other modules notified by an event-message published at the event-bus. This message includes the URI of the resource and will be published at a channel that describes its status.

This workflow implies that some requests cannot be answered using the HTTP 200 code. Instead, the system will answer with the HTTP 202 code indicating that the instruction was accepted and the resource was marked for that operation.

3.4.1.1 Routing-Keys

This module is listening for the following *routing-keys*:

- `topic.created`
- `relation.created`
- `analysis.created`

This module publishes to the following *routing-keys*:

- `source.{created| updated | deleted}`
- `domain.{created| updated | deleted}`
- `document.{created| updated | deleted }`
- `item.{created| updated | deleted }`
- `part.{created| updated | deleted }`
- `word.{created| updated | deleted }`
- `topic.{updated | deleted }`
- `relation.{updated | deleted }`
- `analysis.{updated | deleted }`

3.4.1.2 Use-Case

Let's start a simple example of use of this module. Imagine that a user wants to analyze all the papers published by the *UPM Digital Repository* during year 2014.

So, he/she creates a new *source* doing a POST request on the `/sources/` uri attaching a valid json with the url: `oaipmh://oa.upm.es/perl/oai2?in=2014`.

The api module will create a new *source* with that url and the time filter. After that, it will store it in the internal storage and will publish a new event in the channel: `source.created` with the new uri, e.g `sources/ c941b900-a310-11e5-803e-0002a5d5c51b`.

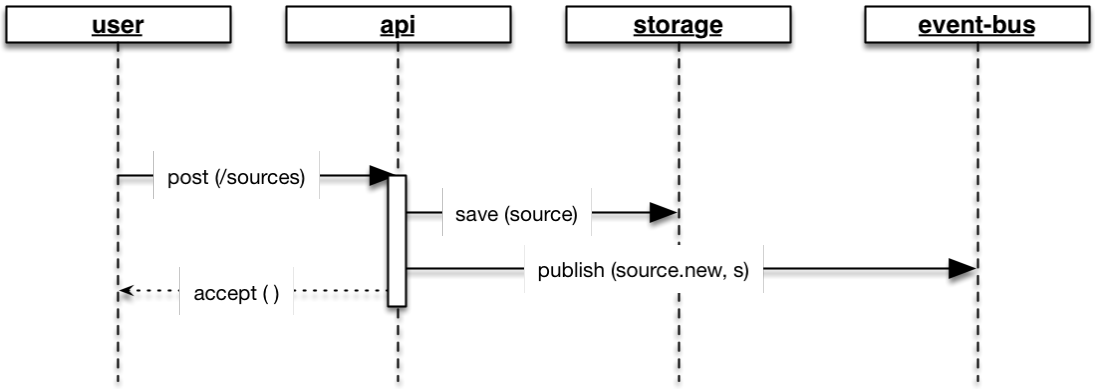

Figure 16. Api Module use-case.

3.4.2 Hoarder

This module is responsible for downloading remote files from *sources* which were previously added to the system by the *Api* module. Thus, it is listening for events published in the event-bus with a routing-key that matches with: '**`source.#`**' .

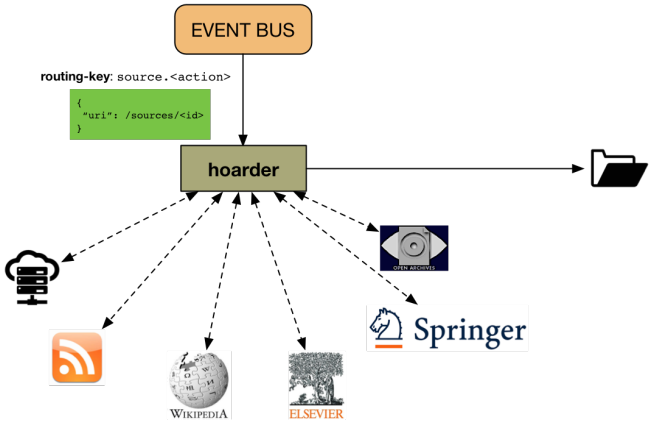

Figure 17. Hoarder module.

Multiple threads are continuously looking for changes in a list of valid *sources*. These *sources* can be remote web services publishing digital content under the OAI specification[28] or following the Web Feed format (e.g rss, atom). In future, they may be journal websites such as Elsevier[29], Springer[30] or even websites as Wikipedia[31]. Moreover, anyone can manually add content to our system using cloud storage services or file-sharing services.

This module can work in synchronous or asynchronous mode:

[28] http://www.openarchives.org/OAI/openarchivesprotocol.html
[29] https://www.elsevier.com
[30] http://www.springer.com/gp/
[31] https://www.wikipedia.org

- In the **asynchronous mode**, it polls the *source* periodically for new content. When a new resource is detected, it will download the file(s) along with the meta-information, if it exists. All these files are moved to a shared folder with the Harvester module. This work mode is typical for sources related to ***publishing services***, e.g. Elsevier, Springer, RSS Server, LAN folders.
- In the **synchronous mode**, it only downloads the data when the new source is added. After that, no more requests are executed on that *source*. This work mode is typical for sources related to ***remote files***, e.g. zip file in Dropbox.

This module has a particular behaviour because it does not publish events to the event-bus, instead it downloads resources to a shared folder with the Harvester module.

At this stage, the resources have not been processed yet. Domain entities are not created until Harvester module process them.

3.4.2.1 Supported Protocols

In the deployed version of *librairy*, we have considered as sources of information both OAI-PMH Data Providers (e.g. *openarchives.org)* and RSS Sites (e.g. *rss.slashdot.org)*. The inherent idea is to force the module to handle different protocol specifications based on common steps typical for data providers. Both RSS and OAI-PMH providers should publish the meta-information using *dublin-core* terms and also publish it in a passive way, i.e. waiting for a previous request defining a temporal window to list all the available resources. However, OAI-PMH Data Providers usually publish related files (pdf, doc, etc) that have to be downloaded separately from the meta-information, while RSS Sites always include the content and the meta-information in the same resource.

Taking into account these features, the first step was to choose an integration framework that enabled us to represent these differences and to reuse the common actions. In Java there are mainly three frameworks: Spring-Integration[32], Mule-Enterprise Service Bus (Mule-ESB)[33] and Apache Camel[34] with these features. All these solutions are known to implement the well-known Enterprise Integration Patterns (EIP)[35] offering a standardized domain-specific language to integrate applications and a set of adapters to support integration with common external services such as FTP, WS-REST, AMQP services and so on.

Taking into account that they all are open-source and they have special features such as error handling, transactions, multi-threading, scalability and monitoring, and so on, the really discriminative aspects between them are the number of supported technologies, the domain-specific language (DSL) used and the facilities to create new adapters. In fact, our hoarder has to communicate with OAI-PMH providers, so we need to develop a new OAI-PMH adapter. For this reason, the procedure to create new adapters is a key feature for us, as well as having a large

[32] http://projects.spring.io/spring-integration/
[33] https://www.mulesoft.com/platform/soa/mule-esb-open-source-esb
[34] http://camel.apache.org.
[35] http://www.enterpriseintegrationpatterns.com

number of existing components easy-to-integrate with other frameworks. Finally, we decided to use Apache Camel[36] as our integration framework to defines routes and workflows.

```
from("oaipmh://oa.upm.es/perl/oai2?delay=60000").unmarshal().jaxb("es.upm.oeg.camel.oaipmh.
    model").filter().xpath("//item/request/set[contains(.,'physics')]").to("mock:result");
```

Figure 18. Camel route for polling the UPM OAI-PMH Data Provider.

3.4.2.1.1 OAI-PMH Camel Component

We have developed a new camel component[37] for polling OAI-PMH Data Providers. It allows to create Camel routes with an OAI-PMH Provider as source and also converts from OAIPMHType to XML format (marshal) or from XML to OAIPMHType format (unmarshal) using the OAI-PMH XML Schema definition[38]. The purpose of this feature is to make it possible to use Camel's built-in expressions for manipulating OAI-PMH messages. As shown in figure 19, an XPath (XML Path Language)[39] expression can also be used to filter the OAI-PMH message.

We have defined some options to customize the behaviour of this component:

- ***delay***: Delay in milliseconds between each poll.
- ***initialDelay***: Milliseconds before polling starts.
- ***userFixedDelay***: Set to true to use fixed delay between pools, otherwise fixed rate is used.
- ***verb***: OAI-PMH command such as ListRecords, ListIdentifiers, Identify.
- ***metadataPrefix***: Specifies the metadataPrefix of the format that should be included in the metadata part of the returned records.
- **from**: Specifies a lower bound for datestamp-based selective harvesting. UTC DateTime value. After first request, this value is updated to current time if no upper bound is defined
- **until**: Specifies an upper bound for datestamp-based selective harvesting. UTC DateTime value

3.4.2.1.2 Hoarder Routes

Once the OAI-PMH Camel component has been developed, it can be used from a camel route to request the list of records to a Data Provider. After that, we need to understand the obtained record to handle the values of the terms and download the related resource/s. As mentioned before, it has to handle term ambiguity and multi-encoding into the same protocol (RSS or OAI-PMH), so it should be adaptable enough for each DC term used. Usually, the exchanged information is encoded as XML, so we also incorporated XML Path Language (XPATH) here to adapt the expressions to read the term values in our routes to handle the heterogeneity showed before. In future cases where JSON will be the encoding used by servers, we will use JSONPATH instead of XPATH to define these expressions.

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

[37] https://github.com/cbadenes/camel-oaipmh

[38] http://www.openarchives.org/OAI/openarchivesprotocol.html#OAIPMHschema

[39] http://www.w3.org/TR/xpath/

A new route should be created for each new *source* handled. So, we designed a wrapper for Camel context using the Groovy language[40] , to facilitate the way to create/modify OAI-PMH and RSS routes. It allows route definition using a Java style, i.e. using the Camel Domain Specific Language (DSL) directly for creating EIP or routes in a fluent builder style. To reduce the url definitions, we included the following namespaces in the root context:

- *oai*: "http://www.openarchives.org/OAI/2.0/"
- *dc*: "http://purl.org/dc/elements/1.1/"
- *provenance*: "http://www.openarchives.org/OAI/2.0/provenance"
- *oai dc*: "http://www.openarchives.org/OAI/2.0/oai dc"
- *rss*: "http://purl.org/rss/1.0/"

Thus, a route for polling an OAI-PMH data provider, create a metadata file for each record and download the related files is:

```
from("oaipmh://oa.upm.es/perl/oai2?initialDelay=1000&delay=60000").
   setProperty(SOURCE_NAME, constant("upm")).
   setProperty(SOURCE_URL, constant("http://oa.upm.es/perl/oai2")).
   to("direct:setCommonOaipmhXpathExpressions").
   setProperty(PUBLICATION_URL, xpath("//oai:metadata/oai:dc/dc:relation/text()",String.class)
     .namespaces(ns)).
   to("direct:retrieveByHttpAndSave")
```

Figure 19. OAI-PMH route for the UPM repository.

And the equivalent for a RSS repository:

```
from("rss:http://rss.slashdot.org/Slashdot/slashdot?" +
   "splitEntries=true&consumer.initialDelay=1000&consumer.delay=2000" +
   "&feedHeader=false&filter=true").marshal().rss().
   setProperty(SOURCE_NAME, constant("slashdot")).
   setProperty(SOURCE_URL, constant("http://rss.slashdot.org/Slashdot/slashdot")).
   to("direct:setCommonRssXpathExpressions").
   to("direct:retrieveByHttpAndSave")
```

Figure 20. RSS route for the Slashdot site.

In both cases, some common *subroutes*, i.e. set of actions, have been used to encapsulate groups of operations such as *direct:setCommonOaipmhXpathExpressions*, *direct:setCommonRssXpathExpressions* and *direct:retrieveByHttpAndSave*.

A route defines flows of works starting in a *from* point and deriving to one or more output flows (*to*). These common flows define common actions grouped in a same starting point (*from*).

[40] http://www.groovy-lang.org/

In such a way, *direct:setCommonOaipmhXpathExpressions* defines the set of expressions, usually XPATH expressions, to extract the values of the Dublin-Core terms from an OAI-PMH record as follows:

```
from("direct:setCommonOaipmhXpathExpressions").
    setProperty(SOURCE_PROTOCOL,
     constant("oaipmh")).
    setProperty(SOURCE_URI,
     simple("http://www.epnoi.org/oaipmh/${property." + SOURCE_NAME + "}")).
    setProperty(PUBLICATION_TITLE,
     xpath("//oai:metadata/oai:dc/dc:title/text()",String.class).namespaces(ns)).
    setProperty(PUBLICATION_DESCRIPTION,
     xpath("//oai:metadata/oai:dc/dc:description/text()",String.class).namespaces(ns)).
    setProperty(PUBLICATION_PUBLISHED,
     xpath("//oai:header/oai:datestamp/text()",String.class).namespaces(ns)).
    setProperty(PUBLICATION_URI,
     xpath("//oai:header/oai:identifier/text()",String.class).namespaces(ns)).
    setProperty(PUBLICATION_URL,
```

Figure 21. XPath expressions to retrieve OAI-PMH values.

In a similar way, the RSS data provider requires a common direct:setCommonRssXpathExpressions as follows:

```
from("direct:setCommonRssXpathExpressions").
    setProperty(SOURCE_PROTOCOL,
     constant("rss")).
    setProperty(SOURCE_URI,
     simple("http://www.epnoi.org/rss/${property."+SOURCE_NAME+"}")).
    setProperty(PUBLICATION_TITLE,
     xpath("//rss:item/rss:title/text()", String.class).namespaces(ns)).
    setProperty(PUBLICATION_DESCRIPTION,
     xpath("//rss:item/rss:description/text()", String.class).namespaces(ns)).
    setProperty(PUBLICATION_PUBLISHED,
     xpath("//rss:item/dc:date/text()", String.class).namespaces(ns)).
    setProperty(PUBLICATION_URI,
     xpath("//rss:item/rss:link/text()", String.class).namespaces(ns)).
    setProperty(PUBLICATION_URL,
     xpath("//rss:item/rss:link/text()", String.class).namespaces(ns)).
    setProperty(PUBLICATION_LANGUAGE,
     xpath("//rss:channel/dc:language/text()", String.class).namespaces(ns)).
    setProperty(PUBLICATION_RIGHTS,
     xpath("//rss:channel/dc:rights/text()", String.class).namespaces(ns)).
    setProperty(PUBLICATION_CREATORS,
     xpath("string-join(//rss:channel/dc:creator/text(),\";\")", String.class).namespaces(ns)).
    setProperty(PUBLICATION_FORMAT,
     constant("htm")).
    setProperty(PUBLICATION_METADATA_FORMAT,
     constant("xml"));
```

Figure 22. XPath expressions to retrieve RSS values.

In addition, other common flows have been developed such as *direct:avoidDeleted*, *direct:saveToFile*, *direct:downloadByHttp* and *direct:downloadByHttpAndSave* to be reused from

high-level routes such as *direct:setCommonRssXpathExpressions* or *direct:setCommonOaipmhXpathExpressions*.

3.4.2.2 Routing-Keys

This module is listening for the following *routing-keys*:

- `source.{created | update | delete}`

This module does not publish to any *channel*

3.4.2.3 Use-Case

Following the use-case started in the *api* module, an event-message will be received by the hoarder module with the uri of the new *source*. Then, it will read the url of the *source* from the internal storage and will compose a valid OAI-PMH request for getting the list of resources in the specific time interval.

For each OAI record received, e.g. `oai:oa.upm.es:17`, it will compose an XML file with that meta-information and will check if contains related files. For each related file, it will download it and both data (meta-information and related files) are moved to the shared folder.

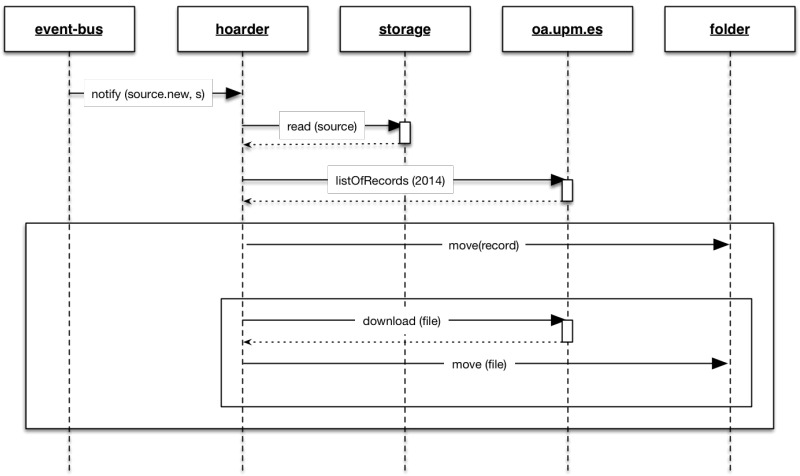

Figure 23. Hoarder module use-case.

3.4.3 Harvester

This module creates domain entities such as *documents, items, parts* and *words*, from both metadata and related files previously downloaded by the *hoarder* module.

Thus, this module is listening for published events in the event-bus with a routing-key that matches with: **`source.*`**.

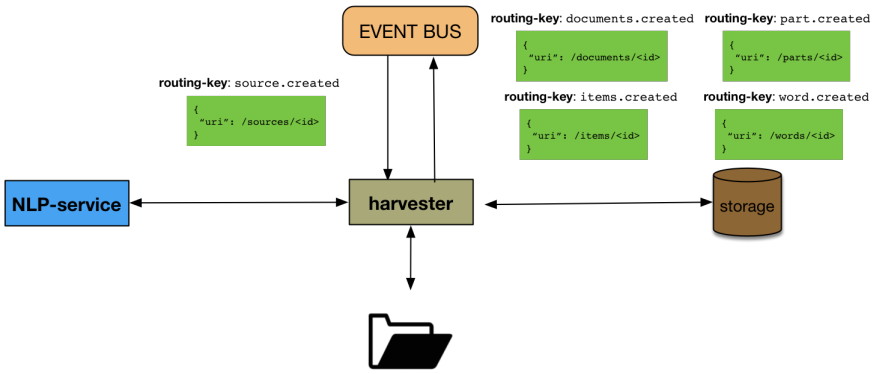

Figure 24. Harvester module.

3.4.3.1 Workflow

This module was also developed using Apache Camel to define the harvesting flows. Now, the goal is to build domain entities containing both meta-information and list of words extracted from the content of the resource or resources. The NLP service is used to extract the text from the pdf files and to obtain the bag-of-words from a resource file. After that, a JSON parser (GSON) is required to create the related domain entity in a predefined JSON scheme.

It consumes files from a folder where the Hoarder module drops them directly, in real time. It is possible using the *doneFileName*[41] option included in Camel that extends the limited functionality defined in the File IO Api of Java JDK. Thus, in a continuous flow of work, the Hoarder downloads resources and meta-information from data providers to a shared folder where the Harvester takes them and generates *documents, items, parts* and *words* from them to be processed by epnoi.

All these resources (i.e. *documents, parts, words* and *parts)* will be stored in the common internal storage and published to the event-bus. The routing-key will include the processed resource and the new status (i.e. *created, updated* or *deleted*).

3.4.3.2 Routing-Keys

This module is listening for the following *routing-keys*:

- `source.{created | updated | deleted}`

This module publishes to the following *routing-keys*:

- `document.{created | updated | deleted}`
- `item.{created | updated | deleted}`
- `part.{created | updated | deleted}`
- `word.{created | deleted}`

[41] http://people.apache.org/~dkulp/camel/file2.html

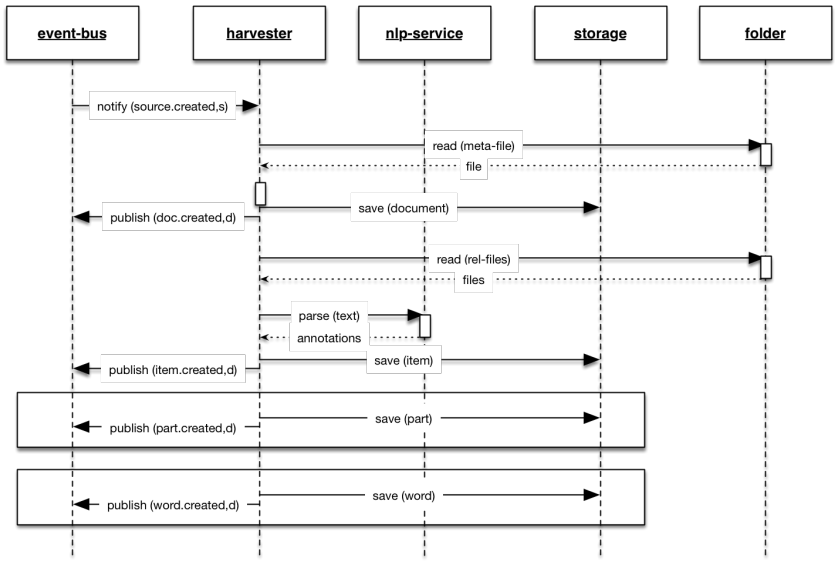

Figure 25. Harvester module use-case.

3.4.3.1 Use-Case

Following with the use-case from the *hoarder* module, the *harvester* module will handle every new file copied to the shared folder. For each file, it will read the meta-information (if exists) and the related files (e.g. pdf paper) to retrieve the text from them.

Then, it will parse the text using the internal NLP service to identify the list of *words* , the logical *parts* and to parse the text content of the *items* as well as to compose a *document* based on the meta-information. For each of them, it will be stored in the internal storage and will be published to the appropriate channel: `document.<status>`, `item.<status>`, `part.<status>` or `word.<status>`.

At this point, more extra information could be even added to *items* or to the *document*, depending on the external semantic resources used such as WordNet[42], Wikidata[43], BabelNet[44] or any other.

3.4.4 Modeler

This module considers the problem of modeling text corpora. The goal is to find short representations of *items,* based on their *parts*, preserving the essential statistical relationships that are useful for tasks such as classification, summarization and similarity and relevance judgments.

Popular algorithms reduce each resource in the corpus to a vector of real numbers taking into account frequencies of words, e.g. The *tf-idf* scheme takes into account the number of occurrences of each word compared to an inverse document frequency count measuring the number of occurrences of a word in the entire corpus. Though they present some appealing features such as

[42] https://wordnet.princeton.edu
[43] https://www.wikidata.org
[44] http://babelnet.org/

the identification of discriminative words, they are unable to reveal inter or intra resource statistical structure.

Other approaches capture most of the variance in the collection and even some aspects of basic linguistic notions, e.g. synonymy and polysemy, such as *Latent Semantic Analysis* (LSA) or *Latent Semantic Indexing* (LSI)[4] but they cannot express more complex relationships between *documents*, between *words* and between *documents* and *words*. Since they are discriminative models, they provide a model only for the target variable, documents, conditional on the observed variables, words, ignoring hidden structures such as topics.

Thus, to describe these *hidden structures*, i.e. these complex relationships, not only a topic model algorithm is required, but also a generative topic model algorithm such as *Latent Dirichlet Allocation* (LDA)[5]. This will enable us to organize and summarize documents at scale, what would be impossible by any other manner, preserving the essential statistical relationships that are useful for the aforementioned tasks. More details about the algorithm used by this module in Appendix 3.

3.4.4.1 Workflow

In this first version of the system, this module can build two types of models:

- A ***topic-model***, using the *Latent Dirichlet Allocation* (LDA) algorithm, for each *domain*. So, a *domain* will be represented by a set of *topics*, i.e. ranked list of words, and the *documents, items and parts* from that *domain* will be represented by a vector of probabilities to belong to these *topics*.
- A ***distributed representational model***, using the Word2Vec (W2V) [9] method, for each *word* in every domain.

As previously mentioned, an LDA model is used to describe the inherent topic distribution of existing *documents*. This model requires some parameters to be built, and they need to be adjusted to obtain a high quality model.

In addition, all parameters are *domain*-level parameters, so we need to calculate new values whenever the *domain* changes. From the point of view of efficiency, this operation is executed in background mode each time a group of *documents* are added. The size of that group is defined beforehand.

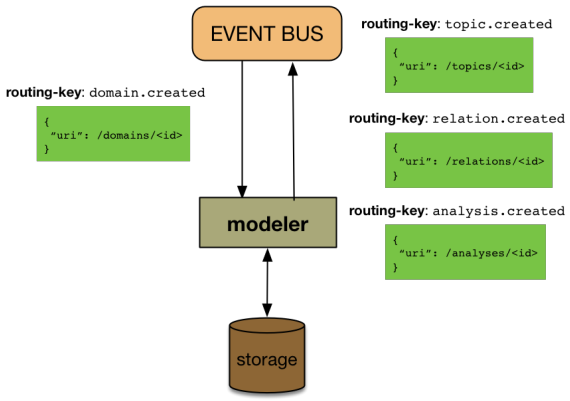

Figure 26. Modeler module.

3.4.4.2 Routing-Keys

This module is listening for the following *routing-keys*:

- `domain.{created | updated | deleted}`

This module publishes to the following *routing-keys*:

- `topic.{created | updated}`
- `relation.{created | updated}`
- `analysis.created`

3.4.4.3 Use-Case

Once an event-message has been received from the event-bus in the channel `domain.created`, this module will read all the *documents* (and/or *items, and/or parts* inside it). Then, it will start to build representational models for that *domain*. For each new *topic* or *relation* discovered, an event-message will be published in the corresponding channel: `topic.created` or `relation.created` with the uri of the resource. When the analysis is done, a new event-message is sent to: `analysis.created` to indicate that the new models have been created and stored in the internal storage along with the *analysis*.

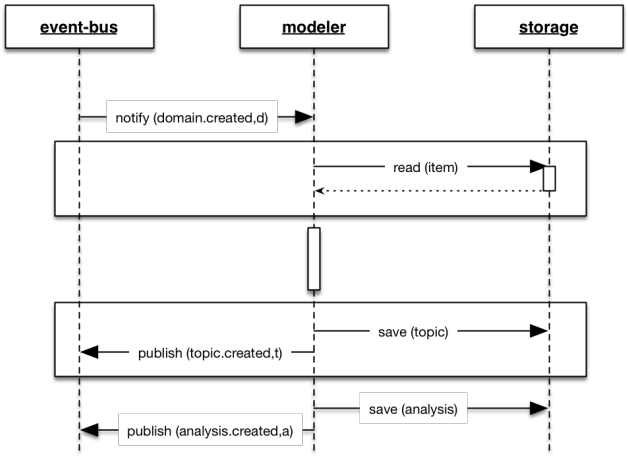

Figure 27. Modeler module use-case.

3.4.5 Comparator

This module use the models created by the *Modeler* module during an *analysis* to calculate the analogies among resources such as *documents*, *items*, *parts* and/or *words.*

As stated previously, the system will discover analogies based on information derived from the *documents, items and/*or *parts*. For this, some metrics are required, mainly similarity measures, to connect them and extract knowledge from these relationships. More details about the similarity measure used by this module in Appendix 3.

3.4.5.1 Routing-Keys

When a *domain* is analyzed, it implies that new *models* are built for each representational level, i.e. domain, *document*, *item*, *part* and *word*. Thus, this module will use these models to measure the similarity value among resources at the same representational level and will update them linking those with a similar value higher than a threshold. So, networks of similar resources are built for each *domain*.

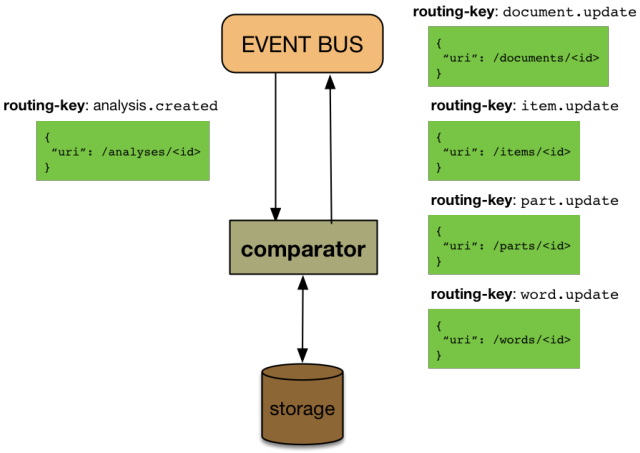

Figure 28. Comparator module.

This module is listening for the following *routing-keys*:

- `analysis.created`

This module publishes to the following *routing-keys*:

- `domain.updated`
- `document.updated`
- `item.updated`
- `part.updated`
- `word.updated`

3.4.5.2 Use-Case

When a new event-message is published in the `analysis.created` channel, this module will read the built model for each resource in that *domain* (i.e. *document*, *item*, *part* and *word*) and will use it to calculate the similarity measure among them.

Once the measure is obtained for each resource, it will update it and will publish an event-message in the appropriate channel, i.e. `document.updated`, or `item.updated`, or `part.updated` or `word.updated`.

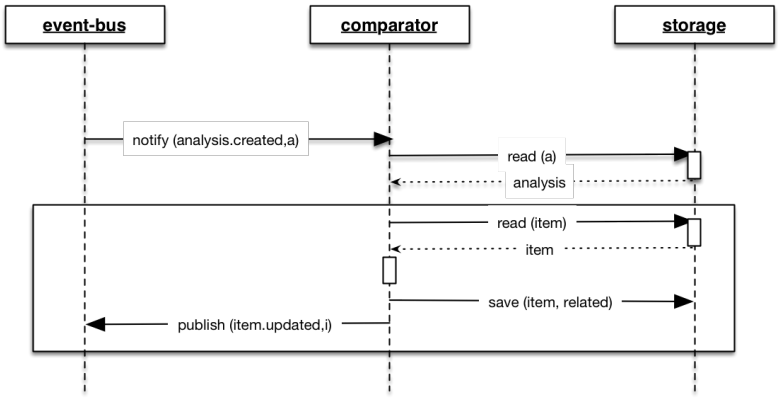

Figure 29. Comparator module use-case.

3.4.6 Learner

This module discovers terms and hypernym relations from the documents of a domain. It mainly interacts with two services, the *Natural Language Processing* (NLP) service to perform the extra annotation tasks not previously executed by the *harvester* module (e.g. dependency parsing) and the *Knowledge-Base* (KB) service, which contains one or more knowledge bases.

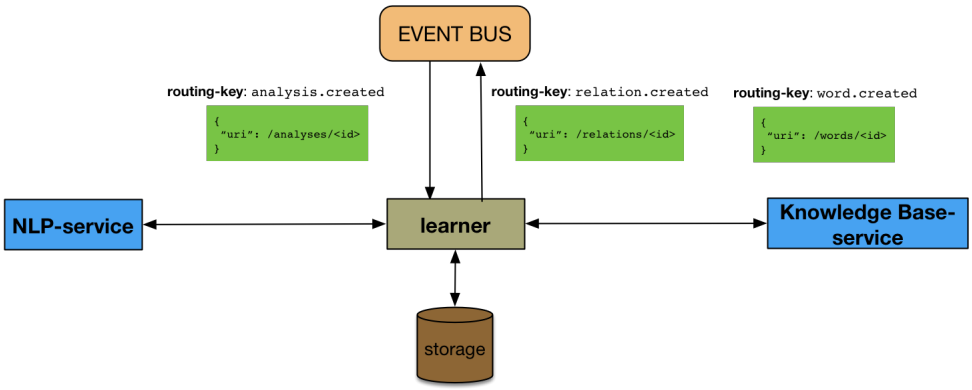

Figure 30. Learner module.

3.4.6.1 Routing-Keys

This module is listening for the following *routing-keys*:

- `analysis.created`

This module publishes to the following *routing-keys*:

- `relation.{created | updated }`
- `word.{created | updated}`

3.4.6.2 Use-Case

When a new notification has been received in the `analysis.created` channel, this module will read all the *items* in that domain to discover *terms* and *relations*. For each new entity recognized, it will publish a new event-message to the `word.updated` or to the `word.created` channel, depending on whether they are an existing *word* or a composition of *words*.

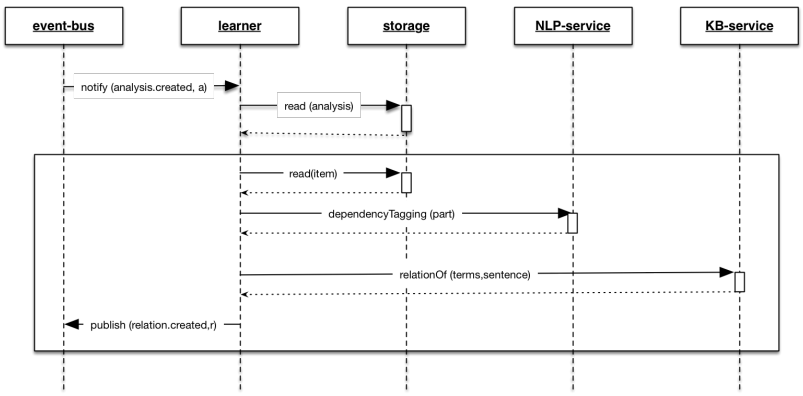

Figure 31. Learner module use-case.

After that, it will gather all the relevant information about the text using the NLP and the KB service to conclude which type of relation exists, or not, between two *terms*. For each new *relation* discovered, a new event-message to `relation.created` or `relation.updated` channel will be published.

3.5 Services

As previously shown in modules section, the system has been designed following the basic principles of the ***Microservices Architecture*** [27]. The main purpose here is not only to have multiple public services, but also to allow building a scalable system in a scalable way.

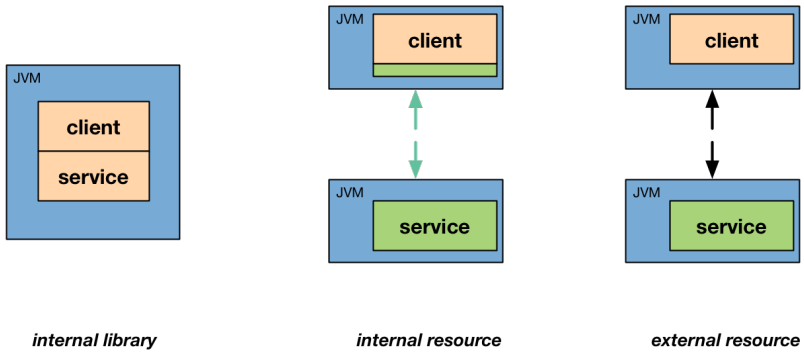

Figure 32. Types of service.

From the *scalable way* point of view, *modules* and *services* are developed and deployed independently of one another. Each of them has its own life-cycle and may be running in an isolated environment.

From the *scalable system* point of view, the architecture is designed applying the *Scale Cube* model. It enables the x-axis scaling, i.e. running multiple copies of a module or a service, the y-axis scaling, i.e. decomposing the system into modules and service, and the z-axis scaling, i.e. running multiple instances of a global installation with different corpus each of them.

3.5.1 Natural Language Processing (NLP)

Some *Natural Language Processing* (NLP) tasks have been externalized as a service to reuse common functionalities and optimize the use of resources. It takes existing NLP libraries such as Gate[45], CoreNLP[46] and the Text Mining Framework[47].

[45] https://gate.ac.uk/

[46] http://stanfordnlp.github.io/CoreNLP/

[47] http://backingdata.org/dri/library/

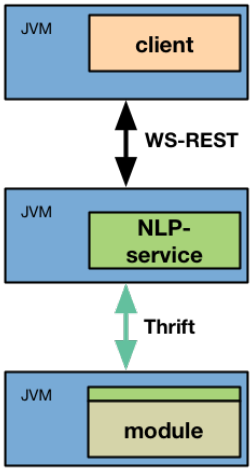

Figure 33. NLP service interfaces.

In short, it offers the possibility of performing the following NLP tasks in a textual document:

- **tokenization**: Splits a stream of text into tokens, i.e. words and symbols.
- **sentence splitting**: Group a sequence of tokens into sentences.
- **lemmatization**: Generates the word lemmas for all tokens (i.e. their cannonical form).
- **stemming**: Reduces the token to the morphological root of the word.
- **part-of-speech**: Labels tokens with their POS tag based on both its definition and its context.
- **entity recognition**: Identifies entities such as Person, Organization, Location, Time and Numerical expressions.
- **Dependency parsing**: Identifies the syntactic relationship that might exist between individual terms within the context of a sentence.

Currently, it is implemented as both an ***internal resource*** and an ***external resource*** having two interfaces: a WS-REST for public clients and a Thrift[48]-based for internal clients.

3.5.2 Knowledge-Base

The knowledge base provides and facilitates a read-only and mostly domain independent curated set of facts that can be considered as the ground truth for other modules such as the Leaner and the Modeler. Currently it uses two well-known knowledge resources:

- **WordNet**. WordNet [28] is a large lexical database of English. Though we are mainly interested in nouns, WordNet also contains information about verbs, adjectives and adverbs. All these elements are grouped into synsets, each expressing a distinct concept (i.e. entities are grouped if they share the same meaning). Synsets are also related between the semantic and lexical relations. In order to access WordNet we use JWI (the MIT Java WordNet Interface)[29], a library that supports fast and compact in-memory access to WordNet versions 1.6 through 3.0, among other related WordNet extensions.

[48] https://thrift.apache.org/

- **Wikidata**. Wikidata is a free linked database that can be read and edited both by humans and machines. Its endeavour is to become a central storage for the structured data of the Wikimedia ecosystem (including Wikipedia). We rely in the Wikidata Toolkit[49] to parse and handle the huge Wikidata dumps, and we create what we refer as a Wikidata view, which contains only the relations that we are interested in for the shake of efficiency. Currently the KB service provides two different Wikidata views, namely:
    - **In memory view.** All the entities and relations of interest contained in the Wikidata dump are loaded to in-memory data structures optimized for fast access.
    - **Unified data manager view.** Since the in memory view option is quite memory demanding, the KB service provides also the possibility of using a version of the Wikidata view stored in a column-oriented database inside the Unified Data Manager. A small latency is introduced, but the hardware requirements are greatly reduced.

As in the case of the NLP service, the KB service is deployed as both an ***internal resource*** and an ***external resource*** having two interfaces: a WS-REST for public clients and a Thrift-based one targeted at local clients where efficiency is a must and no firewalls may block the message interchange.

3.5.3 Storage

Multiples types of data are handled in this system. Sometimes, it should be stored in a key-value form, at other times it should be stored in a column-oriented form. In any case, the system must be flexible enough to handle as different databases as data types it handles.

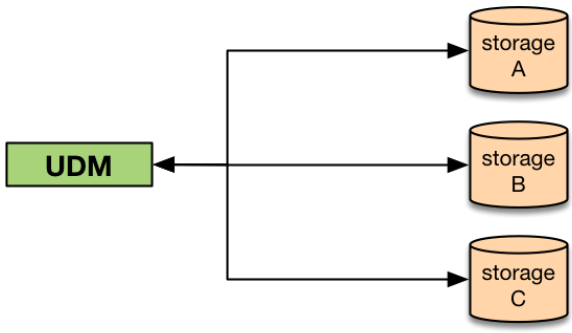

Figure 34. Unified Data Manager.

Inspired in the *Data Access Object* (DAO) pattern, we created a *Unified Data Manager* (UDM) that implements the four basic functions of a persistent storage, i.e. *Create*, *Read*, *Update* and *Delete* (CRUD), for any type of data along with more complex functions combining some of them. In this way, all the particularities of data storage services are only handled from here and any other module has a centralized access for persistent operations.

The overall flow is parallelized by modules, and the storage operations are centralized within each module by the UDM.

[49] https://www.mediawiki.org/wiki/Wikidata_Toolkit

Currently, the following databases are handled in the system:

- **column-oriented database**: Focused on unique and/or structured data. This allows searching key elements in resources. We use *Apache Cassandra* to support this functionality because of its linear scalability and performance.
- **document-oriented database**: Focused on raw data. This allows retrieving all the information gathered about a resource. We are currently working with Apache Solr[50] as text search and analytic engine but, taking into account scability features, we are moving our designs to *Elasticsearch*[51] to allow ***epnoi*** to be deployed in cluster. It is also a wrapper to Lucene with an inbuilt document database but providing scaling solutions. The schema created in Elasticsearch considers only one index ('research') and as type-definitions as resource-types ('documents', 'items', 'parts', 'words','topics','relations', 'source' and 'domains'). In this way, the system will index all resources when they are created.
- **graph database**: Focused on relationships. This allows exploring resources through the relationships between them. We are using OpenLink Virtuoso[52] as our RDF Triple store to evaluate SPARQL queries over resources such as documents, domains, topics and so on.

  Besides oriented to explore the relationships (i.e the analogies among resources) we are using Neo4j[53] as our graph database to allow users to explore the resources using graph algorithms.

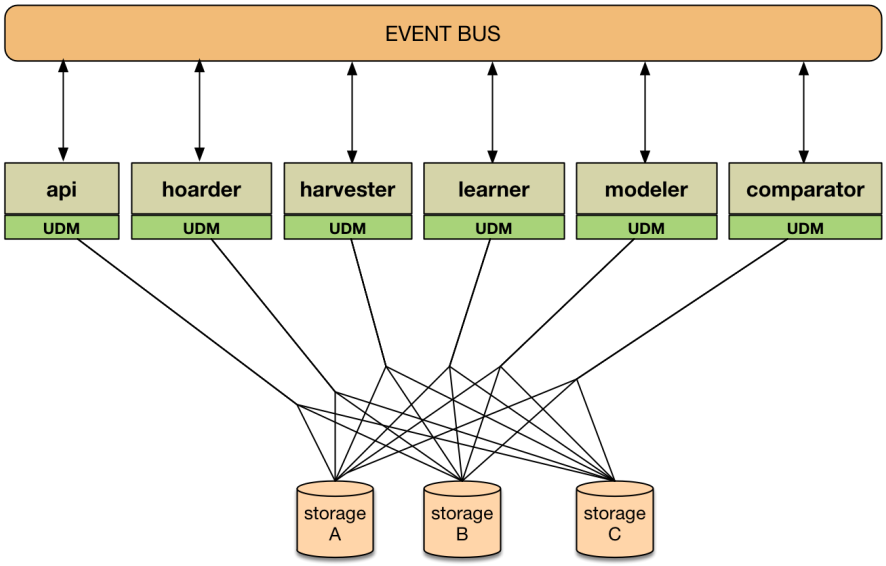

Figure 35. Storage system.

[50] http://lucene.apache.org/solr/
[51] https://www.elastic.co/
[52] http://virtuoso.openlinksw.com/
[53] http://neo4j.com/

4 Application Programming Interface (API)

The system provides programmatic access to the repository content. Deployed version of the API provides a set of functionalities to manage and to explore resources. It allows adding *sources*, grouping *documents* in *domains,* searching *topics*, explore *relations* and so on.

The API follows WS-REST principles and the return format used for all the endpoints is JSON.

Some details about resources and operations are presented below and described in detail in Appendix 4.

4.1 Reading Queries

Detailed information about resources can be obtained using the API endpoints (more details in Appendix 4). This type of queries are considered as *reading queries* such as reading details of a *resource*.

4.1.1 Read details of a Domain

Request:

```
HTTP-GET http://drinventor.dia.fi.upm.es/api/0.2/domains/<domain-id>
```

Request-Example:

```
curl -i -X GET http://drinventor.dia.fi.upm.es/api/0.2/domains/default
```

Response-Example:

```
{
    "uri": "http://drinventor.eu/domains/default",
    "creationTime": "2016-10-11T11:24+0000",
    "description": "attached to source: http://drinventor.eu/sources/default",
    "name": "default
}
```

4.1.2 Read details of a Document

Request:

```
HTTP-GET http://drinventor.dia.fi.upm.es/api/0.2/documents/<document-id>
```

Request-Example:

```
curl -i -X GET
http://drinventor.dia.fi.upm.es/api/0.2/documents/010950f6_4b98_4f32_a1c6_6ccb8c328
6f4
```

Response-Example:

```
{
  "uri": "http://drinventor.eu/documents/010950f6_4b98_4f32_a1c6_6ccb8c3286f4",
  "creationTime": "2016-10-11T13:30+0200",
  "publishedOn": "2012",
  "publishedBy": "siggraph",
  "authoredOn": "2012",
  "authoredBy": "Bruno Galerne, Ares Lagae, Sylvain Lefebvre, George Drettakis",
  "retrievedFrom": "http://drinventor.ccgv.org.uk/db-
siggraph/010950f6_4b98_4f32_a1c6_6ccb8c3286f4/Full.pdf",
  "retrievedOn": "2016-10-11T13:30+0200",
  "format": "pdf",
  "language": "en",
  "title": "Gabor noise by example"
}
```


4.1.3 Read details of a Topic

Request:

```
HTTP-GET http://drinventor.dia.fi.upm.es/api/0.2/topics/<topic-id>
```

Request-Example:

```
curl -i -X GET
http://drinventor.dia.fi.upm.es/api/0.2/topics/5249ce5686b47dd5de4dfb9e18e70e39
```

Response-Example:


```
{
  "uri" : "http://drinventor.eu/topics/5249ce5686b47dd5de4dfb9e18e70e39",
  "creationTime" : "2016-10-11T12:06+0000",
  "content" : "theme,color,image,user,model,rating,figure,face,object,template"
}
```


4.1.4 List Documents in a Domain

Request:

```
HTTP-GET http://drinventor.dia.fi.upm.es/api/0.2/domains/<domain-
id>/documents
```

Request-Example:

```
curl -i -X GET http://drinventor.dia.fi.upm.es/api/0.2/domains/default/documents
```

Response-Example:

```
[
   "http://drinventor.eu/documents/fffcf676_6bd0_4587_8ab4_2ec344a36dc4",
   "http://drinventor.eu/documents/fff58b8d_15f0_4dd6_90b1_7bdba2872141",
   "http://drinventor.eu/documents/ffe44439_a998_4ddd_b30f_6e3a652ebe90",
   ..
]
```

4.2 Searching Queries

Along with the previous *reading queries*, more complex queries based on the textual content of the resources are needed.

For this reason, the system also provides an endpoint to make these type of queries:

filters/<id>/matches

This service is based on the document-oriented storage (Section "Storage), so it allows users to search *documents*, *items* and *parts* indexed in Elasticsearch based on their content.

4.2.1.1 Create a Filter

Request:

```
HTTP-POST http://drinventor.dia.fi.upm.es/api/0.2/filters
```

Request-Example:

```
curl -i -H "Content-Type: application/json" -X POST -d '{"content":"Gabor noise by example"}' http://drinventor.dia.fi.upm.es/api/0.2/filters
```

Response-Example:

```
{
  "uri" : "http://drinventor.eu/filters/21e5d61115cd59b9062b2516a73c588b",
  "creationTime" : "2017-02-28T10:05+0000",
  "content" : "Gabor noise by example"
}
```

4.2.1.2 List matched Items

Request:

```
HTTP-GET http://drinventor.dia.fi.upm.es/api/0.2/filters/<filter-id>/items
```

Request-Example:

```
curl -i -X GET
http://drinventor.dia.fi.upm.es:80/api/0.2/filters/21e5d61115cd59b9062b2516a73c588b/matches/items
```

Response-Example:

```
[ {
```

```
  "weight" : 0.44048798084259033,

  "resource" : "http://drinventor.eu/items/89c38eb9ea20a2828aebc67ce2b9e7d6",

  "description" : "ACM Reference Format\nGalerne, B., Lagae, A., Lefebvre, S.,
Drettakis, G. 2012. Gabor Noise by Exampl.."

} ]
```


4.3 Explorative Queries

Along with the previous *reading and searching queries*, more complex queries based on the relationships among the resources are needed. They are oriented to iterate over the network of resources connected by analogies between them.

4.3.1 From an existing resource

The system provides an endpoint to get the list of similar resources for each resource type (i.e documents, items and parts):

documents/<id>/documents

items/<id>/items

parts/<id>/parts

4.3.1.1 Sorted List of similar Items

Request:

```
HTTP-GET http://drinventor.dia.fi.upm.es/api/0.2/items/<item-id>/items
```

Request-Example:

```
curl -i -X GET
http://drinventor.dia.fi.upm.es/api/0.2/items/c6632036e43a12a63e52c56830d9a292/items
```

Response-Example:


```
[

  "http://drinventor.eu/items/6e9cd55a3f6259d141536cc276023235",

  "http://drinventor.eu/items/3d369b1cc69f0e0449dcf5126238f74",

  "http://drinventor.eu/items/63b730658ccf01bb0576c78ba42f2ec8",

  "http://drinventor.eu/items/40cc66e646ca419856099730544557c2",
…
]
```


4.3.1.2 Similarity score between two Items

Request:

```
HTTP-GET http://drinventor.dia.fi.upm.es/api/0.2/items/<item-id>/items
```

Request-Example:

```
curl -i -X GET
http://drinventor.dia.fi.upm.es/api/0.2/items/c6632036e43a12a63e52c56830d9a292/items/6e9cd55a3
f6259d141536cc276023235
```

Response-Example:

```
{

  "uri" : "http://drinventor.eu/items/c7bf8d0dc3ef40f2bfe371c12814e839",

  "creationTime" : "2016-10-11T12:29+0000",

  "weight" : 0.1273334759947482

}
```

4.3.2 From free-text

Moreover, the system also provides an endpoint to make these type of queries from free-text:

filters/<id>/similar

This service is based on the graph-oriented storage (Section “Storage), so it allows users to search resources iterating on the graph stored in Neo4j.

4.3.2.1 Create a Filter

Request:

```
HTTP-POST http://drinventor.dia.fi.upm.es/api/0.2/filters
```

Request-Example:

```
curl -i -H "Content-Type: application/json" -X POST -d '{"content":"A large form of
digital art, pixel art is created through the use of raster graphics software, where images
are edited on the pixel level. Graphics in most old (or relatively limited) computer and video
games, graphing calculator games, and many mobile phone games are mostly pixel art"}'
http://drinventor.dia.fi.upm.es/api/0.2/filters
```

Response-Example:

```
{

  "uri" : "http://drinventor.eu/filters/827e564428b2589dd1ad31fd5deb85ba",

  "creationTime" : "2017-02-28T10:18+0000",

  "content" : "A large form of digital art, pixel art is created through the use of
raster graphics software, where images are edited on the pixel level. Graphics in most
old (or relatively limited) computer and video games, graphing calculator games, and
many mobile phone games are mostly pixel art"

}
```

4.3.2.2 List similar Items from a Filter

Request:

```
HTTP-GET http://drinventor.dia.fi.upm.es/api/0.2/filters/<filter-id>/items
```

Request-Example:

```
curl -i -X GET
http://drinventor.dia.fi.upm.es/api/0.2/filters/827e564428b2589dd1ad31fd5deb85ba/similar/items
```

Response-Example:


```
[ {
  "weight" : 0.6446552491964215,
  "resource" : "http://drinventor.eu/documents/d36a1c64_30bf_47f8_8732_129e3dc51b10",
  "description" : "Interactive local adjustment of tonal values"
}, {
  "weight" : 0.6441199305756808,
  "resource" : "http://drinventor.eu/documents/b32b2cb9_f759_4838_927f_0c4951553339",
  "description" : "Edge-preserving decompositions for multi-scale tone and detail
manipulation"
}, {
  "weight" : 0.6415959629944215,
  "resource" : "http://drinventor.eu/documents/ed7b785b_3731_423c_879f_8ee136b9be3c",
  "description" : "Color2Gray"
},
…
]
```

5 Installation

The system described in this deliverable could be used as an online service or as a local instance.

5.1 Online Service

The best way to benefit from all the improvements acquired by means of multiple and heterogeneous corpus added to the system by other users is by using the (Swagger) REST-API online service at:

```
http://drinventor.dia.fi.upm.es/api
```

As it includes machine learning functions, the greater the corpus database is, more accurate the results will be.

5.2 Local Instance

There is also a binary distribution of the system that can be installed and run. To facilitate the installation of the system, it has been packaged as Docker[54] containers and uploaded to Docker-Hub[55].

The following images compose the system:

```
librairy/document-db:1.0
librairy/column-db:1.3
librairy/graph-db:1.2
librairy/event-bus:1.0
librairy/explorer:0.21.1
librairy/harvester-research:0.11
librairy/modeler-lda:0.11
librairy/modeler-w2v:0.11
librairy/comparator-jsd:0.2
librairy/comparator-cos:0.2
librairy/ftp:1.0
```

5.2.1 Minimum Recommended Requirements

Hardware Requirements

- *CPU Processor*: four to twelve 64-bit multicore processors.
- *RAM*: 64GB or more.
- *HDD*: at least 500GB of free space.

Software Requirements

- *OS*: Linux, Cloud, Windows and OS X.
- *Docker*: 1.4.1 or newer.

[54] https://www.docker.com/
[55] https://hub.docker.com/

6 References

[1] M. Welsh, “The staged event-driven architecture for highly-concurrent server applications”, *Univ. California, Berkeley*, 2000.

[2] Y. Petinot, C. L. Giles, V. Bhatnagar, P. B. Teregowda, H. Han, and I. Councill, “CiteSeer-API”, *Proc. Thirteen. ACM Conf. Inf. Knowl. Manag. - CIKM ’04*, p. 553, 2004.

[3] R. Wu and D. G. Down, “Round robin scheduling of heterogeneous parallel servers in heavy traffic”, *Eur. J. Oper. Res.*, vol. 195, no. 2, pp. 372–380, 2009.

[4] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, “Indexing by Latent Semantic Analysis”, *J. Am. Soc. Inf. Sci.*, vol. 41, pp. 391–407, 1990.

[5] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent Dirichlet Allocation”, *J. Mach. Learn. Res.*, vol. 3, no. 4–5, pp. 993–1022, 2003.

[6] M. Steyvers, G. Ths, and T, “Probabilistic topic models”, *Landauer, T., McNamara, D., Dennis, S., Kintsch, W., Ed. Latent Semant. Anal. A Road to Meaning. Laurence Erlbaum. Tang, Z. MacLennan, J*, 2006.

[7] T. Hofmann, “Unsupervised Learning by Probabilistic Latent Semantic Analysis”, *Mach. Learn.*, vol. 42, no. 1–2, pp. 177–196, 2001.

[8] A. Asuncion, M. Welling, P. Smyth, and Y. W. Teh, “On Smoothing and Inference for Topic Models”, in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, 2009, pp. 27–34.

[9] T. Mikolov, G. Corrado, K. Chen, and J. Dean, “Efficient Estimation of Word Representations in Vector Space”, *Proc. Int. Conf. Learn. Represent. (ICLR 2013)*, pp. 1–12, 2013.

[10] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, “Multiobjective evolutionary algorithms: A survey of the state of the art”, *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 32–49, 2011.

[11] K. Deb and H. Jain, “An Evolutionary Many-Objective Optimization Algorithm Using Reference-point Based Non-dominated Sorting Approach, Part I: Solving Problems with Box Constraints”, *Ieeexplore.Ieee.Org*, vol. 18, no. c, pp. 1–1, 2013.

[12] K. Deb and R. B. Agrawal, “Simulated Binary Crossover for Continuous Search Space”, *Complex Syst.*, vol. 9, pp. 1–34, 1994.

[13] K. Liagkouras and K. Metaxiotis, “An Elitist Polynomial Mutation Operator for improved performance of MOEAs in Computer Networks”, *Comput. Commun. Networks (ICCCN), 2013 22nd Int. Conf.*, pp. 1–5, 2013.

[14] K. Deb and M. Goyal, “A Combined Genetic Adaptive Search (GeneAS) for Engineering Design”, *Comput. Sci. Informatics*, vol. 26, no. 1, pp. 30–45, 1996.

[15] K. Deb and S. Tiwari, “Omni-optimizer: A generic evolutionary algorithm for single and multi-objective optimization”, *Eur. J. Oper. Res.*, vol. 185, no. 3, pp. 1062–1087, 2008.

[16] V. Rus, N. Niraula, and R. Banjade, “Similarity Measures Based on Latent Dirichlet Allocation”,

in *Computational Linguistics and Intelligent Text Processing*, Springer US, 2013, pp. 459–470.

[17] a Celikyilmaz, D. Hakkani-Tur, and G. Tur, "LDA Based Similarity Modeling for Question Answering", in *Proceedings of the NAACL HLT 2010 Workshop on Semantic Search*, 2010, pp. 1–9.

[18] I. Dagan, L. Lee, and F. C. N. Pereira, "Similarity-Based Models of Word Cooccurrence Probabilities", *Mach. Learn.*, vol. 34, no. 1–3, pp. 43–69, 1999.

[19] R. Ranjan and T. Gneiting, "Combining probability forecasts", *Int. J. Forecast.*, vol. 27, no. 2, pp. 208–223, 2008.

[20] D. Allard, a. Comunian, and P. Renard, "Probability Aggregation Methods in Geoscience", *Math. Geosci.*, vol. 44, no. 5, pp. 545–581, 2012.

[21] V. a. Satopää, J. Baron, D. P. Foster, B. a. Mellers, P. E. Tetlock, and L. H. Ungar, "Combining multiple probability predictions using a simple logit model", *Int. J. Forecast.*, vol. 30, no. 2, pp. 344–356, 2014.

[22] K. T. Frantzi, S. Ananiadou, and J. Tsujii, "The C-value / NC-value Method of Automatic Recognition for Multi-word Terms", *Res. Adv. Technol. Digit. Libr.*, pp. 585–604, 1998.

[23] F. Sclano and P. Velardi, "Termextractor: a web application to learn the shared terminology of emergent web communities", in *Proceedings of the 3th International Conference on Interoperability for Enterprise Software and Applications (I-ESA)*, 2007, pp. 287–290.

[24] M. Mintz, S. Bills, R. Snow, and D. Jurafsky, "Distant supervision for relation extraction without labeled data", *Proc. 47th Annu. Meet. ACL 4th IJCNLP AFNLP*, no. August, pp. 1003–1011, 2009.

[25] R. Snow, D. Jurafsky, and A. Y. Ng, "Learning syntactic patterns for automatic hypernym discovery", *Adv. Neural Inf. Process. Syst. 17*, vol. 17, pp. 1297–1304, 2004.

[26] R. C. Bunescu and R. J. Mooney, "A shortest path dependency kernel for relation extraction", *Proc. Hum. Lang. Technol. Conf. Conf. Empir. Methods Nat. Lang. Process.*, no. October, pp. 724–731, 2005.

[27] B. Costa, P. F. Pires, F. C. Delicato, and P. Merson, "Evaluating REST Architectures - Approach, Tooling and Guidelines", *J. Syst. Softw.*, vol. 112, pp. 156–180, 2015.

[28] C. Fellbaum, "WordNet and wordnets", in *Encyclopedia of Language & Linguistics*, vol. 13, 2005, pp. 665–670.

[29] M. A. Finlayson, "Java Libraries for Accessing the PrincetonWordnet: Comparison and Evaluation", in *Proceedings of the 7th International Global WordNet Conference (GWC 2014)*, 2014, pp. 78–85.

Appendix 1 – Abbreviations and acronyms

DCMI	Dublin Core Metadata Innovation
JSD	Jensen Shannon Divergence
LDA	Latent Dirichlet Allocation
LSA	Latent Semantic Analysis
MOEA	Multi-Objective Evolutionary Algorithm
PLSA	Probabilistic Latent Semantic Analysis
PLSI	Probabilistic Latent Semantic Indexing
OAI-PMH	Open Archive Initiative Protocol for Metadata Harvesting
RDF	Resource Description Framework
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
NER	Named Entity Recognition

Appendix 2 – Learning Algorithm

This section is an advanced text over the deliverable to be submitted for M30, so as to make it available to reviewers.

6.1 Terminology extraction

The initial task of the ontology learning process is terminology extraction, which consists in obtaining the initial domain terms using statistical domain relevance and pertinence indicators. The input for the terminology extraction task is the set of items of the domain terms that belong to the domain terminology. Terminology extraction consists in the following steps:

- **Domain corpus linguistic annotation**. A linguistic annotation process is performed on each of documents that make up the domain corpus, invoking the NLP service. Apart from all the afore described generic NLP tasks, the NLP service also performs what we refer to as term candidates extraction. A transducer is run over each sentence in the documents in the domain corpus in order to find term candidates, which can be seen as noun phrases that structurally can be terms, yet they might not meet the consensus and relevance requirements. The term candidates transducer executes a grammar that formalizes the following noun phrases cases, which are those proposed in [22] that do not harm precision.
- **Termhood calculation**. For each term candidate detected by the transducer in the previous step the Learner calculates its termhood, which is determined by the following measures:
  - **Domain Consensus** [23]. An entropy-related measure that expresses how spread is the use of a certain term throughout the documents of the domain document corpus, since distributed usage expresses a form of agreement and consensus. It is computed as:

$$D_C(t) = -\sum_{d \in D}\left(P(t/d)\log(P(t/d))\right)$$

Where d is any document belonging to the domain D

  - **Domain Relevance** [23]. A probabilistic measure of how relevant a term t is in a domain D with respect to other domains. It is expressed as the probability of the term appearing in the considered domain D divided by the probability of such term in the domain where is it is more likely. Formally it can be expressed as:

$$D_R(t) = \frac{P(t/D)}{\max_k P(t/D_k)}$$

  - **C-value** [22]. This statistical measure is based in the C-value/NC-value [22], since we leave out the contextual part of this measure.

$$C_{value}(t) = \begin{cases} \log_2\|t\| \cdot \#(t) & t \text{ has no superterms} \\ \log_2\|t\| \cdot \left[\#(t) - \frac{1}{|super(t)|}\left(\sum_{st \in super(t)} \#(st)\right)\right] & otherwise \end{cases}$$

Where

- #(t) represents the number of occurrences of the term t in the considered domain.
- ||*t*|| represents the length of the term t.
- *super(t)* is the set of superterms, terms found in the domain in which t is contained.

The termhood of a term candidate is finally determined as the linear combination of the previous measures (those that are not probabilistic are normalized to the [0,1] range). Formally we write:

$$termhood(t) = \alpha norm(Cvalue(t)) + \beta norm(D_C) + \gamma D_R$$

6.2 Relations Extraction

For the extraction of relations we propose the use of a distantly supervised approach, a relation extractor for each type of relations, currently hypernymy relations. The intuition of distant supervision (term coined by [24] but initially proposed by[25]) is that any sentence that contains a pair of entities that participate in a known and curated knowledge base relation is likely to express that relation. It can also be seen as a case of the noisy channel model, in which there is a knowledge base that acts as an oracle that expresses clearly the relationships between knowledge entities, and such information is scrambled thorough the textual domain corpus. Using a distant supervision approach we aim at learning the patterns of how this information is transformed into the sentences where these entities appear. Initially we use [24] intuition and we assume that such relations manifest themselves in a sentence basis (we leave as future line of work the consideration of multiple sentences).

The main advantage of using a distant supervised method is that the training corpus is clearly separated from the domain of appliance corpus. Statistics-based methods, which we apply, need a huge training corpus, it might be possible that the domain corpus were we apply such methods is not big enough. With a distant supervised approach we train our classifiers in a domain-independent manner. The training corpus is created automatically by creating a domain-independent text corpus and leveraging the information contained in available knowledge bases. Once these classifiers have been trained to recognize relations in generic text, these classifiers can be successfully used in the concrete domain corpus, being the system able to extract the domain-specific relations.

The method followed by the Learner for applying a distant supervision approach for hypernym relations extraction is composed by these following steps:

- **Training phase.** Before the Learner can extract relations from a given domain, it must have been previously trained using a large domain-independent corpus of text and one or more knowledge bases used as oracles. Using these elements, we perform the following steps:
  - **Relational sentences corpus creation**. The relational sentences corpus is composed by a large set of sentences where two entities are related. In order to build such corpus in an automatic way:

- We use the periodically available English Wikipedia dumps[56] as our initial English domain-independent textual corpus.
- For each sentence in each of the sections of each wikipedia article, the Learner inspects the entities that appear as term candidates (using the annotations provided by the NLP service) and creates all their possible pairs.
- For each of these pairs, the Learner checks whether there is a hypernymy relationship stored in the knowledge base between these entities making a query to the KB service. If so, the sentence, along with the source and targets of the relation, is added to the relational sentences corpus.

In order to perform such data intensive task, the relational sentences corpus creator system has been implemented on top of the Spark[57] engine for large-scale data processing. A cluster of heterogeneous machines can be easily used to speed up such process.

- **Relational patterns corpus creation**. For each sentence in the training corpus relation patterns can be automatically derived (note that one annotated sentence may generate several relation patterns). Sentences are translated into relation patterns because on the one hand it is helpful to abstract away from concrete subtleties that do not add meaningful information and burden the possibility of its generalization to similar sentences; and on the other hand, to enhance the definitional sentence with additional linguistic information that provides much more information that their simple surface form. For each sentence the Learner generates one or several:
  - **Lexical relational pattern.** The set of lexical hypernym relation patterns are extracted creating a lexical hypernym relation pattern for each sentence in the relational sentences corpus, which implies performing the following actions:
    - We only consider a window of words composed by the words between the source and target of the definition, and with a size W to the left of the source term and to the right of the destination term. The rest of the words in the sentence are removed.
    - The beginning /end of the sentences are marked with a special symbol in case that they belong to the considered window.
    - A linguistic annotation process is performed for each of the considered words in the sentence, and we annotate words with their part-of-speech grammatical category; and the source and destination of the definition are associated with their 7-class Stanford NER (i.e. Date, Location, Money, Organization, Percent, Person, Time)
    - All the words that are not verbs are substituted by their part-of-speech annotation, symbols are left unaltered, and the source/target terms in the definition are replaced by the "<source>"/"<target>" token plus its NER category annotation.

[56] Available at: https://dumps.wikimedia.org/enwiki/

[57] http://spark.apache.org/

- **Syntactic relational pattern.** The set of syntactic hypernym relation patterns are extracted creating a syntactic hypernym relation pattern for each sentence in the relational sentences corpus, which implies performing the following actions:
  - A linguistic annotation process is performed for each of the considered words in the sentence, and we annotate words with their part-of-speech grammatical category; and the source and destination of the definition are associated with their 7-class Stanford NER (i.e. Date, Location, Money, Organization, Percent, Person, Time)
  - A grammatical dependency annotation is performed in the sentence. We remove the directionality of its edges, transforming the sentence into an undirected graph; its nodes are the words plus their part-of-speech annotations, and its arcs binary grammatical relations.
  - Following the Shortest Path Hypothesis [26] all the words that do not belong to the shortest path between the source and target entities are discarded. In case that there is more than one shortest path, a pattern for each path is created.
  - All the words that are not verbs are substituted by their part-of-speech annotation, the source/target terms in the definition are replaced by the "<source>"/"<target>" token plus its NER category annotation.
  - Lastly, as [26] proposes, the negative polarity of verbs must be annotated (i.e. if we have a negation modifier, the prefix '-' is concatenated to the word surface form).

- **Classifier Training**. Once the training corpus of definitional sentences has been transformed into a potentially bigger corpus of hypernym relation patterns, the latter is used to train two classifiers. These classifiers, once trained, will be able to recognize sentences that express hypernym relation between two entities of the sentence. More precisely we perform:
  - **Generative lexical classifier training**. For recognizing the lexical features along with the structure and ordering of definition sentences, we propose a generative approach. We create a probabilistic language model by analyzing the set of lexical hypernym relation patterns that abstracts the training corpus. Once this probabilistic model is built, the generative probability of a given sentence, conditioned to such probabilistic language model becomes the probability of the sentence of being definitional. In order to formalize the language model we use bigram model.

    Once we have trained the probabilistic language model using the set of lexical hypernym relation patterns, we can use the likelihood of new hypernym relation pattern to measure how likely it is to be a definitional sentence that signifies a hypernym relation. In other words, given a sentence form the domain corpus, and two of its terms, the probability of being a definitional sentence is given by the following expression:

$$P_h^l(term_s, term_t, s) = P(hrp^l(term_s, term_t, s) / \lambda)$$

Where

- $hrp^{l}(term_s, term_t, s)$ is a function that given a sentence s and two terms $(term_s, term_t)$, generates the corresponding lexical hypernym relation pattern.
- $P(O/\lambda)$ is the probability of an observation O given the language model λ that we learnt using the training corpus.

- **Discriminative syntactic classifier training**. This classifier is trained using syntactic hypernym patterns from the relational sentences corpus, and once we have removed possible repeated patterns, each of these patterns becomes a feature for the logistic regression classifier. If we extract N distinct patterns, the feature vector would be an N-dimension vector; the i-th position represents whether the i-th syntactic hypernym relation pattern is matched or not. Using the training corpus, the N-dimension feature vector, and a 10-fold cross validation process we train the logistic regression classifier. For the learning algorithm we use Stochastic Gradient Descent (SGD) since it has proved to be an efficient and suitable technique in the context of large-scale learning. The probability of a sentence of being definitional for two terms, from a syntactic perspective, is determined by the following expression:

$$P_h^s(term_s, term_t, s) = \frac{1}{1+e^{-\beta f(s)}}$$

Where:

- β is the regularized weight N+1-dimension vector trained using the SDG
- $f(s)$ is a function that given a sentence s returns an N+1-dimension vector, its i-th element is 1/0 depending on whether it matches the i-th syntactic hypernym relation pattern of the training set.
- $f(s)$ first element is always 1.

- **Extraction phase**. Once the Learner has been trained, it can perform what we refer to as the extraction phase. Given a domain, in the extraction phase the Learner extracts the hypernym relations that appear in textual items of the domain. It translates into checking each sentence of the domain items to determine whether it is a sentence that expresses a hypernym relation or not. If a relation is found a new relation event is generated, containing both the found relation, its estimated probability, and the sentence itself (becoming the hypernym relation provenance sentence). More specifically, for each sentence s_k that belongs to the domain specific corpus the Learner carries out these actions:
  - The set of terms candidates of the sentence are obtained using the very same transducer described for the terminology extraction.
  - The probability of this sentence of being definitional for each pair of terms selected from the terms candidates is calculated as the linear combination of the probabilities obtained using the generative lexical classifier and the discriminative syntactic classifier.

Using the same notation as in the training phase, this probability can be expressed as follows:

$$P_h(term_i, term_j, s_k) = \alpha P_h^s(term_i, term_j, s_k) + \beta P_h^l(term_i, term_j, s_k)$$

In case that this probability is greater than a configurable threshold the hypernym relation is considered as found between these two terms in the given domain; and a created relation event is fired then as a consequence.

Appendix 3 – Similarity Algorithm

This section is an advanced text over the deliverable to be submitted for M30, so as to make it available to reviewers.

6.3 Topic Models

The utility of topic models stems from the property that the inferred hidden structure that they reveal resembles the thematic structure of the collection. A hidden structure is a topic structure such as topics distribution, *per-resource* topic distributions or *per- resource per-word* topic assignments. In turn, ***a topic is a distribution over terms that is biased around those associated under a single theme***. This interpretable hidden structure annotates each resource in the collection and these annotations can be used to deepen the analysis about relationships between resources. In this way, topic modeling provides us an algorithmic solution to managing, organizing and annotating large collections of *documents* according to their *topics*, i.e. according to the distribution of their *words* in *topics*.

Topic modeling algorithms do not require any prior annotations or labelling of the documents. The topics emerge, as hidden structures, from the analysis of the original texts. The main challenge is how to use the observed resources to infer a hidden topic structure.

6.3.1 Latent Dirichlet Allocation

The simplest generative topic model is *Latent Dirichlet Allocation* (LDA)[5]. This and other topic models such as *Probabilistic Latent Semantic Analysis* (PLSA) [7] are part of the larger field of probabilistic modeling. They are well-known latent variable models for high dimensional count data, such as text data in the bag-of-words representation or any other count-based data representation but, while LDA has roots in LSA and PLSA (it was proposed as a generalization of PLSA), it was cast within the generative Bayesian framework to avoid some of the overfitting issues that were observed with PLSA. As mentioned before, since PLSA is a discriminative model, it is unable to describe topics, i.e. hidden structures, but LDA built a generative model to avoid that limitation.

In generative probabilistic modeling, data is treated as arising from a generative process that includes hidden variables. This generative process defines a joint probability distribution over both the observed (O) and hidden random variables (μ). Then data is analyzed by using that joint distribution to compute the conditional distribution of the hidden variables given the observed variables $p(\mu \mid O)$. This conditional distribution is also called the *posterior distribution*. In LDA, the observed variables are the words of the documents, the hidden variables are the topic structure and the generative process is the problem of computing the posterior distribution, i.e. the conditional distribution of the hidden variables given the documents:

$$p(O, \mu) = p(O \mid \mu) \cdot p(\mu) = p(\mu \mid O) \cdot p(O)$$

This statistical model tries to capture the intuition that documents exhibit multiple topics. Each document exhibits the topic in different proportion, each word in each document is drawn from one of the topics, where the selected topic is chosen from the per-document distribution over topics. All

the documents in the collection share the same set of topics, but each document exhibits these topics in different proportion. Documents are each represented as a vector of counts with W components, where W is the number of words in the vocabulary. Each document in the corpus is modelled as a mixture over K topics, and each topic k is a distribution over the vocabulary of W words. Each topic is drawn from a Dirichlet with parameter β, while each document's mixture is sampled from a Dirichlet with parameter α. Formally, a topic is a multinomial distribution over words of a fixed vocabulary representing some concept.

The Dirichlet distribution is a continuous multivariate probability distribution parameterized by a vector of positive reals whose elements sum to 1. It is continuous because the relative likelihood for a random variable to take on a given value is described by a probability density function, and also it is multivariate because it has a list of variables each of whose value is unknown. In fact, the Dirichlet distribution is the conjugate prior of the categorical distribution and multinomial distribution

From a collection of documents, LDA infers: per-word topic assignment, per-document topic proportions and per-corpus topic distributions. Exact inference, i.e. computing the posterior over the hidden variables, for this model is intractable [5], then a variety of approximate algorithms have been proposed [8] such as collapsed Gibbs sampling (CGS), variational Bayesian inference (VB), collapse variational Bayesian inference (CVB), maximum likelihood estimation (ML) and maximum a posteriori (MAP).

Unlike a clustering model, where each document is assigned to one cluster, LDA allows documents to exhibit multiple topics. For example, LDA can capture that one article might be about ”biology” and ”statistics”, while another might be about ”biology” and ”physics”. Since LDA is unsupervised, the themes of “physics”, “biology” and “statistics” can be discovered from the corpus; the mixed-membership assumptions lead to sharper estimates of word cooccurrence patterns.

6.3.1.1 Multi-Objective Evolutionary approach for LDA parameterization

Since LDA is characterized by Dirichlet distributions of topics and documents, i.e. multi- variate generalization of the Beta distribution, it is parameterized by two positive shape parameters, α and β, that appear as exponents of the random variable and control the shape of the distribution. Moreover, the dimensionality of each Dirichlet distribution has to be fixed. So the dimensionality value of the Dirichlet distribution of topics is known and equals to the size of the vocabulary. However, the dimensionality of the Dirichlet distribution of documents, i.e. number of topics, is assumed to be known and fixed.

Thus, we need to estimate three parameters: the number of topics (k), the concentration parameter (α) for the prior placed on documents' distributions over topics and the concentration parameter (β) for the prior placed on topics' distributions over terms . Some authors [8] have proposed inferences to calculate these parameters, however the implementation of LDA made by Spark (based on Expectation/Maximization) and used by **epnoi** does not admit these values yet.

New values of *log-likelihood* and *log-prior* are obtained for each new LDA execution that measures the goodness of the model. The higher these values are, the better the model fits. For this reason, having several conflicting objetives, i.e. improvement of one objective may lead to deterioration of another, and having parameters to estimate (k, α and β), a Multi-Objetive Evolutionary Algorithm (MOEA) is used to find the Pareto optimal solution.

A single solution, which optimizes log-likelihood and log-prior simultaneously does not exist. Instead, the best trade-off solution called Pareto optimal will be obtained. Taking into account performance behaviour [10] and to prevent new objectives derived from the use of the model, the Non-Sorting Genetic Algorithm-III (NSGA-III) [11] is chosen and the optimization problem is defined as follows:

- Objectives:
  - min || log(likelihood) ||
  - min || log(prior) ||
- Constraints:
  - 5.1 < α < 20.0 : Document Concentration. It represents the distribution of a document in topics. That is, how specific is a document. The lower boundary of α (5.1) is greater than the higher boundary of β (5.0) because, in our opinion, if a term belongs to more than one topic, a document will contain equal or more number of topics than those contained in the term. Moreover, we have considered than the number of topics in a document is, at least, 4 times greater than the topics contained in a term, for that reason the higher boundary is 20.0 for α and 5.0 for β.
  - 1.0 < β < 5.0 : Topic Concentration. It represents the distribution of a topic over terms. That is, how a term can belong to several topics. In our opinion, a term can only belong to no more than 5 topics, because greater values will create more ambiguous models.
  - – 0 < k < 2 ∗ p/2 : Number of topics. Usually around the root square of the half of population (p).
- Crossover: Motivated by the success of binary-coded genetic algorithms in prob- lems with discrete search space, the operator selected was Simulated Binary Crossover (SBX) [12] that solves problems having a continuous search space instead of binary. This operator has a search power similar to that of the single-point crossover. It was set to 0.9 to facilitate the explorative capacity of the algorithm.
- Mutation: Mutation operators have been utilized extensively in MOEAs as so- lution variation mechanisms. Mutation operators assist to the better exploration of the search space [13] . Different approaches have been proposed depending on the representation used in MOEAs such as binary or real values. In this case, the operator selected was the Polynomial Mutation operator [14] [15] which allows big jumps in the search space of the decision variable, escaping from local optima and modifying a solution when on the boundary. It was set to 1.0 to promote the explorative analysis.
- Selection: The global best solution is selected by a N-ary Tournament operator. This operator prefers feasible solutions over infeasible solutions (for constraint han- dling), non-dominated solutions over dominated solutions (for handling multiple objetives) and less-crowed solutions over more-crowded solutions (for the mainte- nance of diversity).

The boundaries of the constraints have been defined, as previously mentioned, according to the implementation of the LDA algorithm made by Apache Spark[58]. The minimum value of the parameters, alpha and beta, is defined to 1.0 by default, but they will be different in our system. Since the beta parameter describes the concentration of topics in words, we consider only low values (lower than 5.0 and greater than 1.0) trying to get more representative words for each topic. Defining this range of values, the algorithm will avoid using the same words to characterize different topics, getting distinguished distributions of topics in documents. Moreover, defining high values for the alpha parameter (between 5.1 and 20.0), the algorithm considers that a document can contain more than one topic, but these distributions will not be smooth. We are looking for a characterization that enables us to handle more than one topic in a document, but also enough differences between the topic distributions of different documents to group documents that are talking about the same area or in the same way.

The number of topics will be between 1 and an empirical value defined by the root square of the half of population (p). This approximation is useful to avoid a high exploration during the learning process focusing on a smaller set of values.

6.4 Similarity Measure

Measuring the similarity of *documents* is a key task from which to obtain useful knowledge. Its definition must be general enough to overcome the particular characteristics of the different types of *items* that *documents* aggregate.

Since a *document* contains two kinds of information: context-based, i.e. authors, license rights, format, etc and content-based, i.e. text, image, code,etc the similarity metric should include both concepts.

In short, the measure to calculate the similarity between two *documents* or *items* is a weighted sum of *context-based similarity* (***simctx***) and *content-based similarity* (***simcont***):

$$sim(R_i, R_j) = \alpha * sim_{cont}(R_i, R_j) + (1 - \alpha) * sim_{ctx}(R_i, R_j)$$

where $\alpha \in [0, 1]$

6.4.1 Content-based Similarity

When a *document* is aggregated by other *documents*, the bag-of-words used to calculate the topics distributions is the sum of *words* used in each nested *document*. For this reason, the topics distribution of the root of an aggregation is enough to describe the complete *document*.

Taking into account this premise, the similarity measure between two *documents, items* or *parts* will be based on the distance between their topics distributions. Since they are Dirichlet distributions (probability mass functions), the measure used was the ***Jensen-Shannon divergence***, which can be

[58] http://spark.apache.org/

defined as the average of the *Kullback-Leibler* (KL) divergence between them. KL has two major problems: in the case that one of topics distribution is zero, KL is not defined and it is not symmetric, what does not fit well with semantic similarity measures which in general are symmetric [16]. To solve these problems, Jensen-Shannon divergence considers the average of the distributions as below [17]:

$$JSD(p,q) = \sum_{i=1}^{T} p_i * \log \frac{2 * p_i}{p_i + q_i} + \sum_{i=1}^{T} q_i * \log \frac{2 * q_i}{q_i + p_i}$$

where *T* is the number of topics and *p*, *q* are the topics distributions.

Our similarity measure use the *Jensen-Shannon divergence* transformed into a similarity measure as follows [18] :

$$sim_{cont}(R_i, R_j) = 10^{-JSD(p,q)}$$

where *Ri* , *Rj* are the research entities (i.e. *documents, items* or *parts)* and *p, q* the topics distributions of each of them.

6.4.2 Context-based Similarity

Currently, context-based similarity is only related to author-based similarity. The plan is to include more elements in the future, both extracted directly from the resource or inferred, so as to increase the accuracy of the measure.

But now the system must work properly with authors:

$$sim_{ctx}(R_i, R_j) = sim_{authors}(R_i, R_j)$$

Before obtaining the similarity of authors we need to define how an author (i.e. *user)* is described. Similar to feature vectors to describe the content of a resource, an *author* is represented by a vector that describes adequately his/her most relevant aspects to allow us to take measures between them. The dimension of this vector will depend on the cardinality of the used space.

Based on topic-models, the feature vector will be a multinomial probability distribution so the challenge here is to produce a consensus topics distribution for each author by combining appropriately the topics distributions of their publications. The most popular choice for this aggregation is *Linear Pooling*, which assigns each individual forecast a weight which reflects the importance of the publication, but if we provide an equal weight to every probability the method reduces to an arithmetic average. A Generalized Linear Pooling extends the previous approach considering the possibility of negative weights. However [19] any linear combination of (calibrated) forecast is uncalibrated and lacks sharpness then a *Beta-transformed Linear Pooling* is proposed

applying a Beta transformation to linear pooling operators in order to add a recalibration step to the process and improve their performance. A probability $P_G(A)$ is said to be calibrated if $P(Y_k|P_G(A_k)) = P_G(A_k, k = 1...K)$[19]. Sharpness refers to the concentration of the aggregated distribution. The more concentrated it is, the sharper it is.

Intuitively, aggregation operators based on multiplication seem more appropriate than those based on addition. *Log-linear Pooling* is a linear operator of the logarithms of the probabilities that does not preserve independence and does not verify the marginalization property. *Generalized Logarithmic Pooling* extends it by adding an arbitrary bounded function. On the other hand, instead of establishing a pooling formula from an axiomatic point of view, the aggregation of two distributions could be those that share properties (moments or conditional probabilities) and minimize the KL divergence between them.

Furthermore, as showed in several simulation studies [20], linear pooling performs poorly relative to other pooling formulas with a multiplicative instead of an additive structure. Also, many of non-linear methods involve a large number of parameters, making them computationally complex and susceptible to over-fitting. By contrast, parameter-free approaches, such as the median or the geometric mean of the odds, are too simple to be able to incorporate the use of training data optimally.

Recently, an approach based on the log-odds statistical model of the data has been proposed [21] being an alternative way to express probabilities using the odds ratio.

However, the LDA model considers topics distributions as Dirichlet distribution, i.e continuous multivariate probability distributions , then we can combine them to get a more general topics distribution using the **Bayes' Theorem**.

Thus, considering the following topics distributions (td1,td2) for the research elements (R1,R2) and the topics (T1, T2, T3):

$$td_1 = (t_{11}, t_{12}, t_{13})$$

$$td_2 = (t_{21}, t_{22}, t_{23})$$

and taking into account that:

$$t_{ij} = p(T_i/R_j)$$

the consensus topics distribution *tdf* will be:

$$td_f = (P(T_1/R_1,R_2), P(T_2/R_1,R_2), P(T_3/R_1,R_2))$$

As R_1 and R_2 are independent and using the *Bayes' theorem* we get:

$$P(T_i/R_1, R_2) = \frac{P(R_1) \cdot P(R_2)}{P(R_1, R_2)} \times \frac{P(T_i/R_1) \cdot P(T_i/R_2)}{P(T_i)} = \alpha \times \frac{P(T_i/R_1) \cdot P(T_i/R_2)}{P(T_i)}$$

where α is a class-independent term depending only on the data. As we have measured these data, its value is not interesting here (we are not doing model comparisons), so we treat it as a normalization constant which ensures the Dirichlet constraint :

$$\sum_k P(T_k/R_1, R_2) = 1$$

Now that we know how to combine topic distributions, we can redefine the author similarity (AS) expression using the Jensen-Shannon Divergence as a distance measure of topics distributions and taking its similarity expression for a given interval time:

$$AS_{JSD}(A, B, t_1, t_2) = 10^{-JSD(\hat{a_{12}}, \hat{b_{12}})}$$

where $a_{12}\hat{}$ and $b_{12}\hat{}$ are the consensus topics distributions of authors A and B for the interval of time $t_1 - t_2$.

Appendix 4 – REST API Endpoints

6.5 Sources

A *source* is a repository of documents.

6.5.1 List

List of *sources* in the system.

```
GET http://drinventor.dia.fi.upm.es/api/0.2/sources
```

Request:

```
curl -i -X GET http://drinventor.dia.fi.upm.es/api/0.2/sources
```

Response:

```
200 OK

[
  "http://drinventor.eu/sources/default",
  "http://drinventor.eu/sources/e5d1f9f3e1dd61e0f8f973cf8f960d2a"
]
```

6.5.2 Read

Details about the *source* identified by a URI.

```
GET http://drinventor.dia.fi.upm.es/api/0.2/sources/<source-id>
```

Request:

```
curl -i -X GET
http://drinventor.dia.fi.upm.es/api/0.2/sources/e5d1f9f3e1dd61e0f8f973cf8f960d2a
```

Response:

```
200 OK

{

  "uri" : "http://drinventor.eu/sources/e5d1f9f3e1dd61e0f8f973cf8f960d2a",

  "creationTime" : "2017-02-28T14:12+0000",

  "name" : "bournemouth2016",

  "description" : "Bournemouth OAI-PMH repository in 2016",

  "url" : "oaipmh://eprints.bournemouth.ac.uk/cgi/oai2?from=2016-01-01T00:00:00Z",

  "protocol" : "oaipmh"

}
```

6.5.3 Update

Modify some field of the *source* information identified by a URI and the new one attached to the request.

```
PUT http://drinventor.dia.fi.upm.es/api/0.2/sources/<source-id>
```

Request:

```
curl -i -X PUT  -d '{ "name": "my-new-name"}'
http://drinventor.dia.fi.upm.es/api/0.2/sources/11111111-2222-3333-4444-555555555555
```

Response:

```
204 No Content
```

6.5.4 Create

A new *source* will be created.

```
POST http://drinventor.dia.fi.upm.es/api/0.2/sources/
```

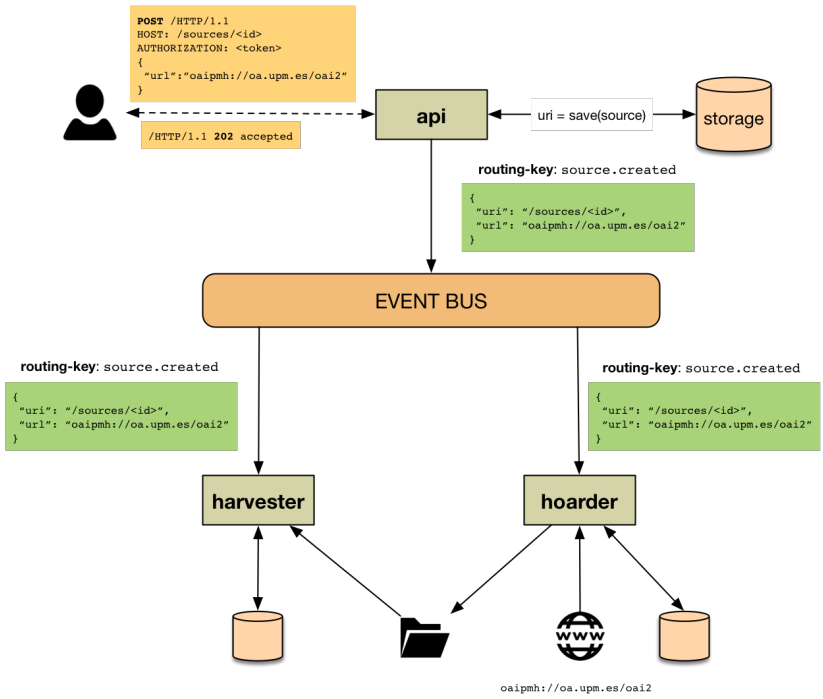

Figure 36. Workflow adding a new source.

The only information required is the *url*.

These are some request examples:

- A OAI-PMH repository:`{ "url": "oaipmh://oa.upm.es/perl/oai2"}`
- An RSS feed:`{"url": "rss://rss.slashdot.org/Slashdot/slashdot"}`

Once the request is accepted by the API module, an HTTP 202 response will be returned indicating that the instruction was accepted and the resource was marked for the creation.

An internal event-message is sent to routing-key *source.created* to publish this action and both the Harvester and the Hoarder module, which are listening for that type of events, will start to make the related operations.

Request:

```
curl -i -X POST -H "Content-Type: application/json" -d '{ "url" :
"oaipmh://oa.upm.es/perl/oai2" }' http://drinventor.dia.fi.upm.es/api/0.2/sources
```

Response:

```
200 Accepted

{

  "uri" : "http://drinventor.eu/sources/2a69499b4ce15fb80f3f832702bc718b",

  "creationTime" : "2017-02-28T15:10+0000",

  "url" : "oaipmh://oa.upm.es/perl/oai2"

}
```

6.5.5 Remove

Delete the *source* identified by a URI.

```
DELETE http://drinventor.dia.fi.upm.es/api/0.2/sources/<source-id>
```

Request:

```
curl -i -X DELETE   http://drinventor.dia.fi.upm.es/api/0.2/sources/11111111-2222-
3333-4444-555555555555
```

Response:

```
204 No Content
```

6.5.6 Remove All

Delete all *source* instances in the system.

```
DELETE http://drinventor.dia.fi.upm.es/api/0.2/sources
```

Request:

```
curl -i -X DELETE  http://drinventor.dia.fi.upm.es/api/0.2/sources
```

Response:

```
204 No Content
```

6.6 Domains

A *domain* is a set of documents. It defines the workspace for the modeler and the learner module. By default, every *source* has a *domain* associated.

6.6.1 List

List of *domains* recorded in the system.

```
GET http://drinventor.dia.fi.upm.es/api/0.2/domains
```

Request:

```
curl -i -X GET  http://drinventor.dia.fi.upm.es/api/0.2/domains
```

Response:

```
200 OK

[
   "http://drinventor.eu/domains/default",
   "http://drinventor.eu/domains/b867903201b89241d168f1f5e5ea1eae",
   "http://drinventor.eu/domains/e5d1f9f3e1dd61e0f8f973cf8f960d2a"
]
```

6.6.2 Read

Details about the *domain* identified by a URI.

```
GET http://drinventor.dia.fi.upm.es/api/0.2/domains/<domain-id>
```

Particular search options can be defined such as *filtering*, *sorting*, *field selection* and/or *paging*, as described in the *Parameters* Section.

Request:

```
curl -i -X GET
http://drinventor.dia.fi.upm.es/api/0.2/domains/b867903201b89241d168f1f5e5ea1eae
```

Response:

```
200 OK

{

  "uri" : "http://drinventor.eu/domains/b867903201b89241d168f1f5e5ea1eae",

  "creationTime" : "2017-02-27T17:01+0000",

  "name" : "isw14",

  "description" : "Papers from the 14th International Semantic Web Conference"

}
```

6.6.3 Update

Update an existing instance of domain.

```
PUT http://drinventor.dia.fi.upm.es/api/0.2/domains/<domain-id>
```

Request:

```
curl -i -X PUT -H "Content-Type: application/json" -d '{ "name": "siggraph" }'
http://drinventor.dia.fi.upm.es/api/0.2/domains/b867903201b89241d168f1f5e5ea1eae
```

Response:

```
204 No Content
```

6.6.4 Create

A new *domain* will be created.

```
POST http://drinventor.dia.fi.upm.es/api/0.2/domains/
```

Request:

```
curl -i -X POST -H "Content-Type: application/json" -d '{ "name": "siggraph",
"description" : "Computer Graphics Corpus" }' http://drinventor.dia.fi.upm.es/api/0.2/domains
```

Response:

```
200 Accepted

{

  "uri" : "http://drinventor.eu/domains/4f56ab24bb6d815a48b8968a3b157470",

  "creationTime" : "2017-02-28T15:24+0000",

  "name" : "siggraph",

  "description" : "Computer Graphics Corpus"

}
```

6.6.5 Remove

Delete the *domain* identified by a URI.

```
DELETE http://drinventor.dia.fi.upm.es/api/0.2/domains/<domain-id>
```

Request:

```
curl -i -X DELETE http://drinventor.dia.fi.upm.es/api/0.2/domains/11111111-2222-3333-
4444-555555555555
```

Response:

```
204 No Content
```

6.6.6 Remove All

Delete all the *domain* instances recorded in the system.

```
DELETE http://drinventor.dia.fi.upm.es/api/0.2/domains
```

Request:

```
curl -i -X DELETE  http://drinventor.dia.fi.upm.es/api/0.2/domains
```

Response:

```
204 No Content
```

6.7 Documents

A *document* is a set of *items* grouped by the same meta-information.

6.7.1 List

List of *documents* recorded in the system.

```
GET http://drinventor.dia.fi.upm.es/api/0.2/documents
```

Particular search options can be defined such as filtering, sorting, field selection and/or paging, as described in the common parameters section.

Request:

```
curl -i -X GET  http://drinventor.dia.fi.upm.es/api/0.2/documents
```

Response:

```
200 OK

[
   "http://drinventor.eu/documents/00106da3_eb2a_4e3c_9f1b_84b1e8c10870",
   "http://drinventor.eu/documents/001a851c_c942_43af_bff3_9edc472cfb14",
   "http://drinventor.eu/documents/00668816_f2c7_462a_a781_9b14c65042eb",
   "http://drinventor.eu/documents/009571ce_1899_4280_bef8_f5003e8ef6b9",
   "http://drinventor.eu/documents/00ee35ca_e662_4432_9298_5797bf43623e",
   "http://drinventor.eu/documents/00f89f8b_935e_46ff_b8f8_20395c43c2a1",
   "http://drinventor.eu/documents/01073f0f_d710_4b4e_961c_725ef5d3f6e8",
   "http://drinventor.eu/documents/010950f6_4b98_4f32_a1c6_6ccb8c3286f4",
   "http://drinventor.eu/documents/019cd23f_fe23_495e_bccf_4ea225ed1877",
…
]
```

6.7.2 Read

Details about the *document* identified by a URI.

```
GET http://drinventor.dia.fi.upm.es/api/0.2/documents/<document-id>
```

Request:

```
curl -i -X GET
http://drinventor.dia.fi.upm.es/api/0.2/documents/01073f0f_d710_4b4e_961c_725ef5d3f6e8
```

Response:

```
200 OK

{

  "uri" : "http://drinventor.eu/documents/01073f0f_d710_4b4e_961c_725ef5d3f6e8",
```

```
  "creationTime" : "2016-10-11T13:30+0200",

  "publishedOn" : "2012",

  "publishedBy" : "siggraph",

  "authoredOn" : "2012",

  "authoredBy" : "Klaus Hildebrandt, Christian Schulz, Christoph von Tycowicz, Konrad
Polthier",

  "retrievedFrom" : "http://drinventor.ccgv.org.uk/db-
siggraph/01073f0f_d710_4b4e_961c_725ef5d3f6e8/Full.pdf",

  "retrievedOn" : "2016-10-11T13:30+0200",

  "format" : "pdf",

  "language" : "en",

  "title" : "Interactive spacetime control of deformable objects"

}
```


6.7.3 Update

Update an existing instance of *document*.

```
PUT http://drinventor.dia.fi.upm.es/api/0.2/documents/<document-id>
```

Request:

```
curl -i -X PUT -H "Content-Type: application/json" -d '{ "title": "Guided Visibility
Sampling", "language" : "en" }' http://drinventor.dia.fi.upm.es/api/0.2/documents/11111111-
2222-3333-4444-555555555555
```

Response:

```
204 No Content
```

6.7.4 Create

A new *document* will be created.

```
POST http://drinventor.dia.fi.upm.es/api/0.2/documents/
```

Request:

```
curl -i -X POST -H "Content-Type: application/json" -d '{ "title": "Interactive
spacetime control of deformable objects", "language" : "en", "retrievedFrom" :
"http://drinventor.ccgv.org.uk/db-siggraph/01073f0f_d710_4b4e_961c_725ef5d3f6e8/Full.pdf" }'
http://drinventor.dia.fi.upm.es/api/0.2/documents
```

Response:

```
200 Accepted

{

    "uri": " http://drinventor.eu/documents/01073f0f_d710_4b4e_961c_725ef5d3f6e8"

}
```

6.7.5 Remove

Delete the *documents* identified by a URI.

```
DELETE http://drinventor.dia.fi.upm.es/api/0.2/documents/<document-id>
```

Request:

```
curl -i -X DELETE http://drinventor.dia.fi.upm.es/api/0.2/documents/11111111-2222-3333-4444-555555555555
```

Response:

```
204 No Content
```

6.7.6 Remove All

Delete all the *document* instances recorded in the system.

```
DELETE http://drinventor.dia.fi.upm.es/api/0.2/documents
```

Request:

```
curl -i -X DELETE  http://drinventor.dia.fi.upm.es/api/0.2/documents
```

Response:

```
204 No Content
```

6.8 Items

An *item* is a textual resource inside a document.

6.8.1 List

List of *items* in the system.

```
GET http://drinventor.dia.fi.upm.es/api/0.2/items
```

Request:

```
curl -i -X GET  http://drinventor.dia.fi.upm.es/api/0.2/items
```

Response:

```
200 OK

[
   "http://drinventor.eu/items/c6632036e43a12a63e52c56830d9a292",
   "http://drinventor.eu/items/6e9cd55a3f6259d141536cc276023235",
   "http://drinventor.eu/items/3d369b1cc69f0e0449dcf5126238f74",
   "http://drinventor.eu/items/63b730658ccf01bb0576c78ba42f2ec8",
   "http://drinventor.eu/items/40cc66e646ca419856099730544557c2",
   "http://drinventor.eu/items/d1fa1030951fab4ee2a0b72c1f2b60bb",
…
]
```

6.8.2 Read

Details about the *item* identified by a URI.

```
GET http://drinventor.dia.fi.upm.es/api/0.2/items/<item-id>
```

Particular search options can be defined such as *filtering*, *sorting*, *field selection* and/or *paging*, as described in the *Parameters* Section.

Request:

```
curl -i -X GET
http://drinventor.dia.fi.upm.es/api/0.2/items/3d369b1cc69f0e0449dcf5126238f74
```

Response:

```
200 OK

{

  "uri" : "http://drinventor.eu/items/3d369b1cc69f0e0449dcf5126238f74",

  "creationTime" : "2016-10-11T11:25+0000",

  "format" : "text",

  "url" : "/librairy/files/downloaded/001a851c_c942_43af_bff3_9edc472cfb14.pdf",

  "content" : "ACM Reference Format\nLai, Y., Hu, S., Martin, R. 2009 …",

  "tokens" : "acm reference format lai martin automatic topology-preserving …"

}
```

6.8.3 Update All

Update the existing instance of an *item*.

```
PUT http://drinventor.dia.fi.upm.es/api/0.2/items/<item-id>
```

Request:

```
curl -i -X PUT -H "Content-Type: application/json" -d '{ "description": "Journal
Paper", "date" : "20060730" }' http://drinventor.dia.fi.upm.es/api/0.2/items/11111111-2222-
3333-4444-555555555555
```

Response:

```
204 No Content
```

6.8.4 Create

A new *item* will be created.

```
POST http://drinventor.dia.fi.upm.es/api/0.2/items/
```

Request:

```
curl -i -X POST -H "Content-Type: application/json" -d '{ "title": "Guided Visibility
Sampling", "format" : "pdf", "description" : "Journal Paper with Conference Talk" }'
http://drinventor.dia.fi.upm.es/api/0.2/items
```

Response:

```
200 Accepted

{

    "uri": "http://drinventor.eu/items/3d369b1cc69f0e0449dcf5126238f74"

}
```

6.8.5 Remove

Delete the *items* identified by a URI.

```
DELETE http://drinventor.dia.fi.upm.es/api/0.2/items/<item-id>
```

Request:

```
curl -i -X DELETE
http://drinventor.dia.fi.upm.es/api/0.2/items/3d369b1cc69f0e0449dcf5126238f74
```

Response:

```
204 No Content
```

6.8.6 Remove All

Delete all the *item* instances recorded in the system.

```
DELETE http://drinventor.dia.fi.upm.es/api/0.2/items
```

Request:

```
curl -i -X DELETE  http://drinventor.dia.fi.upm.es/api/0.2/items
```

Response:

```
204 No Content
```

6.9 Parts

A *part* is a logical section of text in an *item*.

6.9.1 List

List of *parts* recorded in the system.

```
GET http://drinventor.dia.fi.upm.es/api/0.2/parts
```

Request:

```
curl -i -X GET  http://drinventor.dia.fi.upm.es/api/0.2/parts
```

Response:

```
200 OK

[
   "http://drinventor.eu/parts/c6632036e43a12a63e52c56830d9a292",
   "http://drinventor.eu/parts/6e9cd55a3f6259d141536cc276023235",
   "http://drinventor.eu/parts/3d369b1cc69f0e0449dcf5126238f74",
   "http://drinventor.eu/parts/63b730658ccf01bb0576c78ba42f2ec8",
   "http://drinventor.eu/parts/40cc66e646ca41985609973054455 7c2",
   "http://drinventor.eu/parts/d1fa1030951fab4ee2a0b72c1f2b60bb",
…
]
```

6.9.2 Read

Details about the *part* identified by a URI.

```
GET http://drinventor.dia.fi.upm.es/api/0.2/parts/<part-id>
```

Request:

```
curl -i -X GET  http://drinventor.dia.fi.upm.es/api/0.2/parts/11111111-2222-3333-4444-555555555555
```

Response:

```
200 OK

{

  "sense": "discussion",

  "description": "Evaluation and Comparison of results"

}
```

6.9.3 Update

Update the existing instance of a *part*.

```
PUT http://drinventor.dia.fi.upm.es/api/0.2/parts/<part-id>
```

Request:

```
curl -i -X PUT -H "Content-Type: application/json" -d '{ "sense": "conclusion", "description" : "Evaluation of the results" }' http://drinventor.dia.fi.upm.es/api/0.2/parts/11111111-2222-3333-4444-555555555555
```

Response:

```
204 No Content
```

6.9.4 Create

A new *part* will be created.

```
POST http://drinventor.dia.fi.upm.es/api/0.2/parts/
```

Request:

```
curl -i -X POST -H "Content-Type: application/json" -d '{ "sense": "method", "description" :
"Description of the solution" }' http://drinventor.dia.fi.upm.es/api/0.2/parts
```

Response:

```
200 Accepted

{

    "uri": "http://drinventor.dia.fi.upm.es/api/0.2/parts/11111111-2222-3333-4444-
555555555555"

}
```

6.9.5 Remove

Delete the *part* identified by a URI.

```
DELETE http://drinventor.dia.fi.upm.es/api/0.2/parts/<part-id>
```

Request:

```
curl -i -X DELETE   http://drinventor.dia.fi.upm.es/api/0.2/parts/11111111-2222-3333-
4444-555555555555
```

Response:

```
204 No Content
```

6.9.6 Remove All

Delete all the *part* instances recorded in the system.

```
DELETE http://drinventor.dia.fi.upm.es/api/0.2/parts
```

Request:

```
curl -i -X DELETE  http://drinventor.dia.fi.upm.es/api/0.2/parts
```

Response:

```
204 No Content
```

6.10 Words

A *word* is a *term* or an *entity* (i.e. person, organization and place) or any other meaningful unit of text contained in a *part* and/or an *item*.

6.10.1 List

List of *words* recorded in the system.

```
GET http://drinventor.dia.fi.upm.es/api/0.2/words
```

Request:

```
curl -i -X GET  http://drinventor.dia.fi.upm.es/api/0.2/words
```

Response:

```
200 OK

[
   "http://drinventor.eu/words/9624e87065736f776bf6c4ce5db7ab0e",
   "http://drinventor.eu/words/82d65a575279b463b10fbb964ee0f7a9",
   "http://drinventor.eu/words/f484570d7cf557020e11ace406901b10",
   "http://drinventor.eu/words/b891b62ab9be7813b9c97aec94a62fff",
   …
]
```

6.10.2 Read

Details about the *word* identified by a URI.

```
GET http://drinventor.dia.fi.upm.es/api/0.2/words/<word-id>
```

Particular search options can be defined such as *filtering*, *sorting*, *field selection* and/or *paging*, as described in the *Parameters* Section.

Request:

```
curl -i -X GET
http://drinventor.dia.fi.upm.es/api/0.2/words/9624e87065736f776bf6c4ce5db7ab0e
```

Response:

```
200 OK

{

  "uri" : "http://drinventor.eu/words/9624e87065736f776bf6c4ce5db7ab0e",

  "creationTime" : "2016-10-11T12:06+0000",

  "content" : "stroke"

}
```

6.10.3 Update

Update the existing instance of a *word*.

```
PUT http://drinventor.dia.fi.upm.es/api/0.2/words/<word-id>
```

Request:

```
curl -i -X PUT -H "Content-Type: application/json" -d '{ "content": "multi-element"}'
http://drinventor.dia.fi.upm.es/api/0.2/words/11111111-2222-3333-4444-555555555555
```

Response:

```
204 No Content
```

6.10.4 Create

A new *word* will be created.

```
POST http://drinventor.dia.fi.upm.es/api/0.2/words/
```

Request:

```
curl -i -X POST -H "Content-Type: application/json" -d '{ "content": "pixel"}'
http://drinventor.dia.fi.upm.es/api/0.2/words
```

Response:

```
200 Accepted

{

    "uri": " http://drinventor.dia.fi.upm.es/api/0.2/words/11111111-2222-3333-4444-
555555555555"

}
```

6.10.5 Remove

Delete the *word* identified by a URI.

```
DELETE http://drinventor.dia.fi.upm.es/api/0.2/words/<word-id>
```

Request:

```
curl -i -X DELETE   http://drinventor.dia.fi.upm.es/api/0.2/words/11111111-2222-3333-
4444-555555555555
```

Response:

```
204 No Content
```

6.10.6 Remove All

Delete all the *word* instances recorded in the system.

```
DELETE http://drinventor.dia.fi.upm.es/api/0.2/words
```

Request:

```
curl -i -X DELETE  http://drinventor.dia.fi.upm.es/api/0.2/words
```

Response:

```
204 No Content
```

6.11 Topics

A *topic* is a set of *words* related in a *domain.*

6.11.1 List

List of *topics* recorded in the system.

```
GET http://drinventor.dia.fi.upm.es/api/0.2/topics
```

Request:

```
curl -i -X GET  http://drinventor.dia.fi.upm.es/api/0.2/topics
```

Response:

```
200 OK

[
    "http://drinventor.eu/topics/7cfeec00ca747794685ef9a2c30a9544",
    "http://drinventor.eu/topics/f3f9a7cf1ee2c74376a4f544332c18c5",
    "http://drinventor.eu/topics/9c40b8ee2eaea488fd9a45f23643c841",
…
]
```

6.11.2 Read

Details about the *topic* identified by a URI.

```
GET http://drinventor.dia.fi.upm.es/api/0.2/topics/<topic-id>
```

Particular search options can be defined such as *filtering*, *sorting*, *field selection* and/or *paging*, as described in the *Parameters* Section.

Request:

```
curl -i -X GET
http://drinventor.dia.fi.upm.es/api/0.2/topics/7cfeec00ca747794685ef9a2c30a9544
```

Response:

```
200 OK

{

  "uri" : "http://drinventor.eu/topics/7cfeec00ca747794685ef9a2c30a9544",

  "creationTime" : "2016-10-11T12:06+0000",

  "content" : "path,figure,shape,surface,sample,function,image,lens,model,point"

}
```

6.11.3 Update

Update the existing instance of a *topic*.

```
PUT http://drinventor.dia.fi.upm.es/api/0.2/topics/<topic-id>
```

Request:

```
curl -i -X PUT -H "Content-Type: application/json" -d '{ "content":
"path,figure,shape,surface"}' http://drinventor.dia.fi.upm.es/api/0.2/topics/11111111-2222-
3333-4444-555555555555
```

Response:

```
204 No Content
```

6.11.4 Create

A new *topic* will be created.

```
POST http://drinventor.dia.fi.upm.es/api/0.2/topics/
```

Request:

```
curl -i -X PUT -H "Content-Type: application/json" -d '{ "content":
"path,figure,shape,surface"}' http://drinventor.dia.fi.upm.es/api/0.2/topics
```

Response:

```
200 Accepted

{

    "uri": " http://drinventor.dia.fi.upm.es/api/0.2/topics/11111111-2222-3333-4444-
555555555555"

}
```

6.11.5 Remove

Delete the *topic* identified by a URI.

```
DELETE http://drinventor.dia.fi.upm.es/api/0.2/topics/<topic-id>
```

Request:

```
curl -i -X DELETE   http://drinventor.dia.fi.upm.es/api/0.2/topics/11111111-2222-3333-
4444-555555555555
```

Response:

```
204 No Content
```

6.11.6 Remove All

Delete all the *topic* instances recorded in the system.

```
DELETE http://drinventor.dia.fi.upm.es/api/0.2/topics
```

Request:

```
curl -i -X DELETE  http://drinventor.dia.fi.upm.es/api/0.2/topics
```

Response:

```
204 No Content
```