Fusepool P3

Fusepool P3 – Fusepool Publish–Process–Perform Platform for Linked Data

FP7 – ICT – 2013 – SME – DCA (609696)

Deliverable 5.2

Data retrieval for semantic enrichment / processing and semantic indexes for domain specific data retrieval

D5.2 Lead:	Reto Gmür (BUAS / CH)
Author:	Reto Gmür (BUAS)
1st Quality Reviewer:	Milos Jovanovik (OGL)
2nd Quality Reviewer:	Andrey Bratus (SPAZIO)
Coordinator's Contact Person:	Thomas Gehrig (BUAS/CH)
Deliverable nature:	Report (R)
Dissemination level (Confidentiality):	Public
Contractual delivery date:	2015, June 30 (M18)
Actual delivery date:	2015, July 9
Version:	1.0
Total number of pages:	12
Keywords:	Data retrieval

Table of Contents

1. Document History

Version	Name	Date	Remark
1.0	Reto Gmür	09.07.15	Documentation after agreement

2. Documentation Information

Deliverable Nr Title: D5.2 Data retrieval for semantic enrichment/processing and semantic indexes for domain specific data retrieval

Lead: Reto Gmür (BUAS)

Author: Reto Gmür (BUAS)

Publication Level: Public

3. Document Context Information

Project (Title/Number): Fusepool P3 (609696)

Work Package / Task: WP5 / T5.3 & T5.4

Official Citation: Fusepool-P3-D5.1

4. Acronyms and Abbreviations

Acronym	Description
API	Application Programming Interface
DoW	Description of Work
LDP	Linked Data Platform
LDPC	Linked Data Platform Container
PHP	PHP hypertext Preprocessor
RDF	Resource Description Framework
REST	Representational State Transfer
SPARQL	SPARQL Protocol and RDF Query Language

5. EXECUTIVE SUMMARY

This deliverable covers the interaction of Fusepool P3 with external non-human resources and actors. It is thematically and technologically adjacent on one side to D4.1 which covers the interaction with human users and which uses the services provided by this deliverable and on the other side with D5.3 which provides the generic data storage/access RDF API and is a constituent part of the P3 Platform Architecture (D5.1).

The data retrieval for semantic enrichment supports both components delivering the data to the platform, notably GUI components allowing to upload files as well as components harvesting data from external sites. The data retrieval is tightly integrated with the data transformation components provided by Work Package 2 which uses the same RESTful RDF API as the components performing the semantic enrichment provided by Work Package 3.

The components involved are tightly integrated in that they provide a single entry points hiding away all the complexity of the various processing steps. On the other hand the components are loosely coupled which does not just improve stability and flexibility, but also greatly enhance debugging possibilities by having a very small set of well defined and monitorable points of interactions. Throughout the refinement, enrichment and processing life cycle the data passes interfaces defined by the P3 Transformer API that allow to monitor the state of the processing so that corrective measures can be taken if needed.

With T5.4 this deliverable comprises the provision of application level retrieval and query services. As specified by the Platform Architecture wherever possible the interaction between data consuming and the data-storing components shall follow standard protocols and interaction patterns, most notably the ones defined by the Linked Data Platform Specification [LDP]. The architecture foresees T5.4 components to allow some application specific access interfaces provided by services interacting directly with the underlying triple store as opposed to using the standard interactions APIs. As shown by D4.1 the generic API did not needed to be extended with non-standard custom interfaces to accommodate the requirements of the client components. In one case however it was necessary to switch from the originally envisaged LDP interaction model to the SPARQL query language and protocol as this offered a far better performance. The fact that no specific interfaces needed to be introduced proves the versatility of the chosen standard protocols and greatly increases flexibility as well as protection of the investments of the organizations deploying the P3 platform as it allows them to freely choose between standard compliant backends.

6. INTRODUCTION

The purpose of this deliverable it to provide both for the data retrieval, i.e. allow to retrieve the data that shall be semantically enriched, as well as for the services providing domain/application level views on datasets.

The data retrieval for semantic enrichment supports both components delivering the data to the platform, notably GUI components allowing to upload files as well as components harvesting data from external sites. The components delivering content to the platform use standard interaction mechanisms to add contents to a Transforming LDPC as this has been defined in the Platform Architecture D5.1. While D5.1 describes the functional role as well as the protocols supported by the P3 Proxy this report shall focus on the implementation of this crucial mediator between data retrieval and the data processing leading to queryable semantic content. The Transforming LDPC architecture presupposes an agent adding data to the container. Such an agent can be a GUI component allowing the user to add files to an LDPC of the workspace or the crawler that retrieves remote resources at regular intervals.

On the side of the application level retrieval and query service the architecture of the platform foresees the possibilities of services providing a custom HTTP API that interact directly with the triple store. As written in the D5.1 report, such services are only implemented if the development of the UI shows that an interaction via the standard mechanism SPARQL and LDP is not feasible or not providing an adequate level of performance. We will see that a change of the standard protocol could avoid the introduction of such service, increasing the portability of the platform across storage backends.

7. TASKS

This deliverable covers the following tasks:

T5.3 – Data retrieval for semantic enrichment: The Fusepool P3 platform will provide workspaces for the semantic enrichment process. This ensures that the original state of the data is kept throughout the enrichment process and enables storage and retrieval of meta-information from the semantic enrichment process (e.g. to backtrace processing steps leading to (un)wanted linking results).

T5.4 – Application level retrieval and query services: provide domain/application level views on datasets including specialized query services. This includes spatial/temporal queries as applied by YAGO2 as well as text/label based retrieval methods. For the later open-source search technologies like Apache Solr and elasticsearch customize for RDF dataset will be exploited.

8. POSITIONING WITHIN THE PROJECT LIFE SPAN

The DoW has foreseen this deliverables for project month 9. However the chosen architecture described in D5.1/5.3 emphasizes on using pre-existing standards wherever possible and writes about the domain specific query API and services foreseen by T5.4 (one of the two tasks comprising this deliverable) that "such services are only implemented if the development of the UI shows that an interaction via the standard mechanism SPARQL and LDP is not feasible or not providing an adequate level of performance". Defining and implementing such service at an early project stage would be a typical example of premature optimization. It was thus beneficial to postpone this task to a late stage in the project so that the application specific services could be defined to address empirically observed rather than be based on merely hypothized problems.

As for the other task of this deliverable T5.3 recognizing that the chosen modular architecture offers a variety of interfaces suitable for adding the required debugging and backtracing possibilities we opted to develop the required tools based on the concrete requirements that arise during the product development. As an unforeseen dividend of the standard compliance we found that standard tools can be used and no development was necessary. This shows that it was a good decision not to develop tools beforehand, which would have not only been futile, but also an unnecessary addition of complexity to our software corpus.

As postponement of both task of this deliverable was indicated in our view, we consequently decided to postpone this deliverable.

9. DATA RETRIEVAL

9.1 Proxy

The functional role and relevant APIs for the LDP Transforming Proxy are defined in D5.1. This deliverable report shall provided some more details on how the proxy has been implemented and how it is used.

The P3 Proxy is a *reverse proxy* (https://en.wikipedia.org/wiki/Reverse_proxy), i.e. it appears to the client as ordinary sever, clients are not aware of the proxy, what they see is an LDP implementation supporting the P3 Transforming Container extension. The proxy is *transparent* (https://en.wikipedia.org/wiki/Proxy_server#Transparent_proxy) to the backend as well as to the client it never modifies neither the client's request nor the server response. As it forwards the HTTP `Host` header unmodified the backend might technically notice the presence of a proxy as the port indicated by the value of this header may not match the port the backend is actually bound to. Using reverse proxies is very common in productive environment and as such implementation typically process HTTP messages independently of the network binding this should not be an issue for any LDP implementation. However, for completeness D5.1 specifies that the LDP implementation must not assume that the requested IRI reflects the port it is listening to. Practically the LDP implementation will take the public address of proxy as its own public address (hostname and port).

The Marmotta LDP implementation has the ability to automatically configure itself on the first request based on the `Host` header, it assumes this to be the canonical address of the instance and redirect all subsequent requests to use this hostname and port, because of this, it is important that the first request is made via the proxy and using the correct public facing hostname (i.e. not using a hostname like `localhost` if the instance shall later be accessed from the internet). Unlike a classical file serving HTTP server for an LDP implementation it is important to consider the full URI the client is dereferencing as the LDP resources are identified by their full URI and not just by the URI-path section of the request. The HTTP responses of the LDP implementation encoding RDF normally describe the requested resource and contain its URI, it is thus important that the URIs are used consistently.

The proxy doesn't do anything but forwarding the semantically unmodified request and response (with a byte-by-byte identical message body) for all HTTP requests methods but HTTP POST. In the case of HTTP POST it does the following in addition to forwarding the unmodified request:

- It dereferences the resource against which the request is directed with an HTTP GET request, and with an Accept header indicating preference for RDF representations
- If an RDF representation of the resource is returned check if it says that the requested resource is a transforming LDPC
- If it is a transforming LDPC get the URI of the transformer from the graph and use the transformer client library to transform the body of the POST request.
- If and when transformation succeeds, POST the request to the backend keeping all headers of the original requests except those which contradict the properties of the transformation results (e.g. Content-type) and the Slug header which is modified by adding a suffix. The inclusion of the original headers allows the proxy to work in setting where the backend requires authentication.

9.2 Crawler

The integration of the Virtuoso Crawler is described in detail in D2.1 and shall not be repeated here. The crawler is a service to efficiently retrieve data for storage and semantic enrichment with the P3 platform. While building on the established Virtuoso Crawler it now supports the LDP standard for storing the data with the P3 platform.

9.3 Monitoring data

Originally we planned to implement a logging no-operation transformer for debugging to provide the traceability envisaged by T5.3. This no-op transformer could have been added at any position of a transforming pipeline to log the data passed from one transformer to the next. The no-op transformer was thought to be useful to monitor the data at each of its transformation steps. Experience has shown however that standard TCP/IP network monitoring tools where sufficient in most cases. The only case we encountered where standard TCP/IP monitoring tool were not powerful enough was when debugging a scenario where different pipelines that are run concurrently access the same transformer, but even in this case we did not need the planned logging no-operation transformer but could use a standard logging HTTP Proxy such as The Grinder [Grinder], given that the transformer API is based on HTTP and the RDF data exchange bases on standard serialization formats no further tool was needed. This finding clearly exemplifies the benefits of the chose standards based architecture with loosely coupled components communicating via HTTP and standard format.

9.4 Dashboard

The Dashboard allows the user to visually interact with transforming containers (defined in D5.1). The dashboard supports multiple configurations effectively allowing the user to have multiple visual workbenches to serve different usage scenario of one or of several users. With the Dashboard the user can drag-and-drop files to the platform, as defined by the Transforming Container API these files are not only added to the container, but a transformation job is also started in the background. The user will immediately see and be able to access the uploaded data (i.e. the file drag-and-dropped to the visual representation of the container and shortly after the transformation completed also the results of the transformation. The dashboard is described in detail in D4.1.

10. APPLICATION LEVEL RETRIEVAL

With T5.4 the DoW foresees the development of application level retrieval and query services. The platform architecture described in D5.1 stipulates that interaction with the platform by non-human agents as well as interactions between platform components shall be based on either established protocols and standards for interacting with linked data, specifically:

- Linked Data Platform 1.0 - LDP
- SPARQL 1.1 Query Language - SPARQL

As well as protocols defined by the P3 project. The protocols defined within the project shall follow the REST design principles. Notably the following protocols where defined within P3:

- Transformer API
- Transforming Container API
- Transformer Registry API
- User Interaction Request API

Apart from there adherence to the REST principles and (except for the first) relying on the LDP specification what these protocols have in common is that the payload of the messages (the HTTP message body) is expressed using a media type encoding an RDF graph (or, where the LDP specification foresees this, a serialization that will become an RDF graph once the LDP instance assigns the necessary base IRI for parsing). To foster interoperability at the semantic level the platform architecture also defines which ontologies shall be used for the purpose of describing semantic enrichments, notably:

- Open Annotation [Sanderson2013]
- Fusepool Annotation Model (FAM)

These standards and API describe generic and domain independent data retrieval and processing mechanism. The use of standard ontologies for describing annotations allows for text/label based retrieval using the standard SPARQL protocol. Where the backend supports GeoSPARQL [Perry] it is also possible to run more advanced topological queries using the SPARQL protocol. It should however be noted that simple bounding-box based spatial/temporal queries can be run efficiently against any SPARQL endpoint without requiring specific extensions.

The expected price of genericity is a degradation of performance as well as a higher complexity required by the client software. While the services developed within T5.4 should if possible also follow the overall design patterns, i.e. provide RESTful interfaces with RDF payloads they should be designed for specific domains and applications. This would allow to optimize performance as well as to provide the simplest possible API for the use case at hands. As written earlier we would be guilty of premature optimization if we had defined such APIs before having concrete empirical facts on shortcomings of the generic APIs. The necessity and the content of the services implemented withing T5.4 thus depends on the results of T4.1 which shall identify possible optimizations of the generic APIs based on the development of clients.

10.1 Findings from T4.1

As described in report for D4.1 the generic APIs could readily be used and tested in the creation of the UI components. In some cases however the UI implementation was changes to use the platform's SPARQL endpoint rather than accessing the data via LDP as originally foreseen and implemented, this is discussed in some more details in the next section. Only some minor adaptations to the generic APIs where conducted based on the expirience with the UI development.

- Because of a limitation in PHP is is not possible to send an HTTP response with status code 202 and a location header, therefore it was not possible to implement spec conformant implementation of asynchronous transformers using the PHP programming language. Because of this the Transformer API was modified to allow the use of a 201 status code in place of the semantically more accurate 202 status code. The client library was adapted to support this novelty. As the novelty describes only an optional and its usage discourage no adaptation of the transformer implementation library for Java was needed.
- As many clients are implemented in JavaScript and running in the browser the same-origin policy [SPO] applies. This policy is a security mechanism implemented by all modern browsers preventing scripts in a web page to access data and services on servers other than the "origin" server which server the web page. Luckily with the Cross-Origin Resource Sharing (CORS) Specification [CORS] there is a standard mechanism that allows services to override the default policy and be accessible even from scripts running in pages served from different hosts. As with CORS an existing standards is available and nothing in the Restful APIs we defined is incompatible with this specification, no adaptation in our specification was needed. We simply changed the transformer library and all transformer to add support for CORS. We also addressed some issues with the CORS support in Marmotta. The P3 Proxy needed no adaptation as it forwards any CORS support the proxied service provides.

10.2 LDP vs. SPARQL

The User Interaction Request API as well as the other registries (Transfomer Registry and Transformer Factory Registry) describe how regular LDP capabilities are used to maintain these registries. While LDP access to the registries is efficient for adding new entries as well as for browsing from one entry to the next it is inefficient when retrieving a large amount of entries including their description. Using LDP this requires at least one HTTP request to get the index and an additional request for every entry. As the used LDP implementation provide the data of the RDF resources also via the SPARQL endpoint some UI components now access the registry via SPARQL. As D5.1 already specified that all data in the RDF Graphs managed by a Fusepool P3 platform backend can be accessed via SPARQL, no change to the specification was considered necessary.

11. CONCLUSIONS AND FUTURE WORK

At first glance the results of this deliverable might seem disappointing, not only because of the late delivery. First the task descriptions are only somehow loosely connected to what was effectively foreseen to implement under the tasks of this deliverable. This inexact matching lies in the differences between the architecture presupposed by the DoW and our architectural choices. The DoW seems to assume that the state after individual processing steps might somehow get lost or at least become hard to access inside the application and that special mechanism must be provided for backtracing and debugging; this is indeed a very reasonable concern for more classical monolithic applications. In the planning of the application with the chosen architecture the debugging component became a transformer that - from an application point of view - is indistinguishable from the other transformers. The application specific APIs became a possible fastlane trading in genericity and portability for performance. Finally we ended up implementing nothing under the tasks of this deliverable as both artifacts turned out to be superfluous.

As software development is not an exact science we clearly could have avoided this null-result. Nothing prevented us from implementing the foreseen T5.3 component right away and to provide a domain specific query API optimized for a concrete backend at an early stage of the project. While this would have served the psychological and social motives opposing null results and leading to the well known publication bias it would have been detrimental to the software to which unnecessary complexity would have been added as well as to the research findings of this project. Yes, we did not write a single line of code even though we were expected to. But what's more important is that we could show that pervasive and strict adherence to standards and a modular loosely coupled architecture provides some very concrete benefits, reducing complexity and implementation costs.

> *«In my opinion, perfection is not reached when there is nothing left to add but when there is nothing left to remove.»*
>
> ***Antoine de Saint-Exupéry***

Ref.	Description
CORS	http://www.w3.org/TR/cors/
LDP	http://www.w3.org/TR/ldp/
Grinder	http://grinder.sourceforge.net/g3/tcpproxy.html
Perry	Perry, M., Herring, J. (2012). OGC GeoSPARQL - A Geographic Query Language for RDF Data. Open Geospatial Consortium. http://www.opengeospatial.org/standards/geosparql
Sanderson2013	Sanderson, R., Ciccarese, P., Van deSompel, H. (2013). Open Annotation Data Model. Community Draft, W3C. http://www.openannotation.org/spec/core/
SPO	http://www.w3.org/Security/wiki/Same_Origin_Policy
SPARQL	http://www.w3.org/TR/sparql11-query/(http://www.w3.org/TR/sparql11-query/)