Fusepool

STREP

FP7 – 296192

D2.1 Architecture and standards specified and cloud platform and components running

D2.1 Lead: Andrew Janowczyk (Searchbox)

Authors: Reto Gmür, Sarven Capadisli, Andrew Janowczyk, Luigi Selmi

1st Quality reviewer: Anton Heijs / Daniël van Adrichem

2nd Quality reviewer: Michael Kaschesky

Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	30 June 2013
Actual delivery date:	5 July 2013 (updated 30 Dec 2013)
Version:	1.2
Total number of pages:	30
Keywords:	Fusepool, information retrieval, user requirements, living lab, software architecture, data mining, linked data, social curation, machine learning, graphical user interface, data visualization, user engagement, hackathon

Abstract

This report for Deliverable D2.1 covers work tasks T2.1, T2.2, T2.3 and T2.4. The necessary documentation, reports and implementations, in brief, for the tasks are as follows:

T2.1: Defined a cloud software architecture which can support the Fusepool platform in a high-availability, big data, high speed infrastructure. Here we specify the schematics which create a lower bound for a rapidly reconfigurable, scalable and highly adaptive system for different load situations.

T2.2: Implement the plan created for T2.1. As cloud computing is associated with a monthly premium for availability, the team has agreed to use a development environment for all collaborative efforts (with developers maintaining their own local versions as necessary). A migration plan from the development environment to a production environment, once development has been completed, is exactly specified and included.

T2.3: Implement an automated import task for ingesting information from key identified data sources such as Pubmed, disease information and patents. As the data is imported, it is processed to generate the mapping to ontology at schema and instance levels. All necessary data is then persisted for later high speed retrieval.

Executive summary

This work package is the most critical, but also the easiest to review, due to its supporting role of the front facing technology. If the backend isn't setup correctly, then everything else will fail. The tasks in this WP are the design and implementation of a cloud based infrastructure which can support the needs of the Fusepool team. The Fusepool platform is based on established open source projects as well as the results of the IKS EU Research Project. Basing on existing open source projects not only allows us to incorporate a lot of existing code but also leads to an interaction with existing communities helping to divulge the Fusepool project and its code and ensure its sustainability after the funding period.

Once that infrastructure is in place, the absorption of various data sources, analysing them and making them searchable became the next challenge we surmounted. Thankfully due to the expertise of all of the partners, who were specifically chosen for their experience and knowledge of their assigned tasks, the various tasks were completed without any hindrance. We know this to be the case because today we can go online and see the preliminary versions of our search engine running, serving information, and indexing documents. For a high level review of the tasks, please feel free to review the previous page which contains the abstract and a summary of the deliverables.

Document information

FP7 Project Number	296192	**Acronym**	Fusepool
Full Title	Fusepool – Information pooling for product/service development and research		
Project URL	http://www.fusepool.eu		
Document URL			
Project Coordinator	Michael Kaschesky		
EU Project Officer	Martina Eydner		

Deliverable	**No.**	D2.1	**Title**	Architecture and standards specified and cloud platform and components running
Work Package	**No.**	WP2	**Title**	Data sourcing and component integration

Date of Delivery	**Contractual**	M12	**Actual**	M12
Status	Version 1.2		Final X	
Nature	Prototype □ Report X Dissemination □			
Dissemination level	Public X Consortium □			

Authors (Partner)	Reto Gmür, Sarven Capadisli, Andrew Janowczyk, Luigi Selmi			
D2.1 Lead	**Name**	Andrew Janowczyk	**E-mail**	andrew.janowczyk [at] searchbox.com
	Partner	Searchbox	**Phone**	+41 78 914 88 02

Version Log			
Issue Date	**Rev. No.**	**Author**	**Change**
10-6-2013	I	A. Janowczyk	Initial Version
13-6-2013	II	A. Janowczyk	Updated Diagrams
19-6-2013	III	A. Janowczyk	Inserted Reto Gmür's contribution titled "Authentication and authorization in the Fusepool platform" in section T2.1
24-6-2013	IV	L. Selmi	Inserted BUAS contribution titled "Data access and gathering with basic data curation" in section T2.3
3-7-2013	V	A. Janowczyk	Inserted Reto Gmür's contribution titled "Ongoing Platform Development" in section T2.2
4-7-2013	VI	A. Janowczyk	Inserted Luigi Selmi's update of BUAS contribution in T2.3 section
4-7-2013	VII	A. Janowczyk	Integrated William Darling's reviewer feedback
5-7-2013	VIII	A. Janowczyk	Integrated Daniël van Adrichem's reviewer feedback
13-1-2014	IIX	R. Gmür	Modified according to the review report
27-1-2014	IX	R.Gmür	Integrated Michael Kaschesky's reviewer feedback
29-1-2014	X	R.Gmür	Integrated Daniël van Adrichem's reviewer feedback

Table of Contents

Abbreviations

Acronym	Full meaning
API	Application programming interface
BUAS	Bern University of Applied Sciences
DoW	Description of Work (part of the Grant Agreement)
ENOLL	European Network of Living Labs
GA	Grant Agreement
GEOX	Geox KFT
LL	Living Lab
MS	Milestone
OSGi	Open Services Gateway initiative
RAD	Rapid application development
RDF	Resource Description Framework
SEARCH	Searchbox SA
SME	Small and Medium Enterprise
SPARQL	SPARQL Protocol and RDF Query Language
SWOT	Strengths, Weaknesses, Opportunities, and Threats
TREPA	Treparel Information Solutions BV
WP	Work package
XEROX	Xerox SAS
XSLT	eXtensible Stylesheet Language Transformations
XML	Extensible Markup Language

1. Introduction

As is well known, infrastructure and architecture often determine the success or failure of projects. This work package is specifically geared towards not only designing a robust system but also for its implementation. Thankfully, the efforts required to complete this work package were not unexpectedly difficult because we could piggy back on the established OSGi componentization technologies and previous projects that have already combined them with semantic web standards. This only left the task of designing a high-availability (HA) system. In the following subsections we briefly review the exact requirements and introduce the reader to the specific milestones that required completing. Afterwards, each additional section describes in detail the process by which things have evolved to meet the necessary specifications.

1.1. Software architecture, standards and components (Task T2.1)

The work on the 'Privacy preserving data gathering and analysis' concerns Task T2.1 in the DoW which this deliverable D2.1 reports on. No major changes to the task specification were required when carrying out the task.

This task was broken down into several subtasks with subtask-deadlines that were defined and agreed in the first meeting of the Fusepool General Assembly on 25 July 2012 about three weeks after the Fusepool kick-off meeting. The sub-tasks are defined as follows:

- T2.1a – Stanbol, architecture specification [6/2013 planned; 6/2013 actual]
- T2.1b – Cloud subcontracting criteria [6/2013 planned; 6/2013 actual]

1.2. Cloud platform and component implementation (Task T2.2)

The work on the 'Cloud platform and component implementation' concerns Task T2.2 in the DoW which this deliverable D2.1 reports on. No major changes to the task specification were required when carrying out the task.

This task was broken down into several subtasks with subtask-deadlines that were defined and agreed in the first meeting of the Fusepool General Assembly on 25 July 2012 about three weeks after the Fusepool kick-off meeting. The sub-tasks are defined as follows:

- T2.2a – Stanbol-based platform running in development [10/2012 planned; 10/2012 actual]
- T2.2b – Showcases with KMX, searchboxAPI, GeoxNER [11/2012 planned; 11/2012 actual]
- T2.2c – KMX, searchboxAPI, GeoxNER as Stanbol enhancers running (minimal viable product MVP) [1/2013 planned; **4/2013 actual**]
- T2.2d – Stanbol-based platform with LINK GUI portlet [3/2013 planned; **5/2013 actual**]

1.3. Data access and gathering with basic data curation (Task T2.3)

The work on the 'Data access and gathering with basic data curation' concerns Task T2.3 in the DoW which this deliverable D2.1 reports on. No major changes to the task specification were required when carrying out the task.

This task was broken down into several subtasks with subtask-deadlines that were defined and agreed in the first meeting of the Fusepool General Assembly on 25 July 2012 about three weeks after the Fusepool kick-off meeting. The sub-tasks are defined as follows:

- T2.3a – Showcases with common data (patent, opportunity, partner) [8/2012 planned; **9/2012 actual**]
- T2.3b – Data integration Patent DBs, FP7 funding, PubMed [12/2012 planned; **5/2013 actual**]

2. Software architecture, standards and components (Task T2.1)

The most critical aspect of the platform, especially in early planning stages, is choosing a suitable framework and creating a schematic description which software pieces exist and how they interact. This framework determines the interaction and the API definition of the components delivered by the project partners as well as which components have to be developed within the project and where we can rely on existing solutions.

The components of the framework are integrated within an open source project available on GitHub[1], components that fit into one of the underlying software shall be donated to the respective open source communities.

2.1. Background

The main goal of this task was to provide a framework, or at least a lower bound, describing the minimum resources required for a production ready system. By providing the best possible description, we can ensure that the developers, and thus the planners, have a knowledgeable view of the specifications for the system. Without such a task, there could be no concrete agreement between parties on the necessary pieces required for instantiating their systems in a co-operative manner.

2.2. Used technologies and tools

The following basic technologies, software tools, and architectural pattern were chosen. Some of the architectures are entailed by other choices (i.e. transitive dependencies), we only list such tools where they play a crucial role, or require some major interaction with the software we develop or are so fundamental that our evaluation of them was necessary before we could decide to use the tools depending on them.

RDF Data Storage an Exchange

The DoW references the RDF data format on several occasions and because the RDF data model is the lingua franca of the linked data web, it is the obvious choice for the storage of the structured data of the Fusepool Platform. As the Fusepool Platform is designed to be itself part of the Linked Data Web, the RDF format is also exposed and consumed by services so that most of the exchange between the Fusepool Platform, its clients, and providers takes place as serialized RDF.

REST Architecture

The REST architecture[2] is essentially the architecture of the World Wide Web that makes it scalable and resilient. A REST API is based on a hypertext driven mechanism allowing users or software agents to discover and use informational resources. In practice the term is often used more broadly for any HTTP Service relying on the HTTP methods as opposed to remote procedure calls. While this broader definition is used by several components, for our own development we shall stick to the narrower and more versatile definition of REST leading to self-describing services.

OSGi component architecture

While the REST architecture allows interaction of services running on hosts spread over the internet implemented using any programming language, OSGi[3] specifies a component architecture for the local interaction within a Java Virtual Machine. OSGi promotes modularization of the application providing a model for versioned libraries as well as service architecture. The Fusepool platform relies on the core OSGi specification as well as on the Declarative Service specification of the OSGi compendium[4]. The latter allows

[1] https://github.com/fusepool

[2] REpresentational State Transfer (REST) is an architectural style, defined by Dr. Roy Fielding's PhD thesis, *Architectural Styles and the Design of Network-based Software Architectures*. See http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[3] http://www.osgi.org/

[4] http://www.osgi.org/download/r4v42/r4.cmpn.pdf

it to easily build services extending platform functionality. OSGi also allows distribution over multiple machines, which provides and additional layer of scalability in case the foreseen REST based load balancing shouldn't suffice.

Apache Maven[5]

Maven is a software build automation tool. Unlike other more imperative build automation took like Make or Ant, it describes the structure of the code and specifies which plugins will be used while building the software artifact. Maven takes care of retrieving all build time dependencies of a project. This means that a developer can just compile the Fusepool Platform without having to install all required libraries manually. Maven supports a hierarchical project structure so that the Fusepool platform itself consist of several modules which are in turn Maven projects. Maven brings the componentization and dependency management at compile time that OSGi brings at runtime. So each maven project typically builds one OSGi bundle.

Apache Felix

Apache Felix[6] is the used OSGi container. It is one of the tree major open source OSGi containers. Its usage within commercial systems such as the Adobe CQ5 content management system as well as being the foundation of Oracle's GlassFish application service show that Felix is a mature solution ready for application requiring high standards in terms of performance and stability. Besides using the OSGi container the Fusepool Platform is using serveral components provided by the Apache Felix project:

- Service Component Runtime: The Felix implementation of the declarative service specification;
- HTTP Service: An implementation of the OSGi HTTP service using the Jetty HTTP Server[7];
- Web Console: A Web based management console;
- Maven Bundle Plugin: A Maven plugin to create OSGi bundles with maven;
- Maven SCR Plugin: A Maven plugin to create declarative services using Java annotations.

Apache Lucene and Solr

Lucene[8] and Solr[9] are fast and scalable indexing and search tools running on the Java platform. The basic search and indexing are provided by Lucene. Solr frontends Lucene with an HTTP interface. We have chosen Lucene as search solution because of its good performance in terms of search speed and quality of the results, it's existing integration with Clerezza and Stanbol (which uses Solr), as well as the expertise partner SearchBox already had with this tool.

Apache Clerezza

Clerezza[10] is a framework and toolkit for building linked data applications. It provides an API for accessing and manipulating RDF data which is very closely oriented to the W3C RDF specification and is independent on any particular triple store. It provides adapters so that many triple stores can be used for the actual data storage. The Fusepool Platform uses Clerezza for dealing with RDF data as well as its Composite Resource Index Service (CRIS) which allows to use Lucene for indexing structures of an RDF Graph.

[5] Apache and all apache project names are either registered trademarks or trademarks of the Apache Software Foundation in the United States and/or other countries.

[6] http://felix.apache.org/

[7] http://www.eclipse.org/jetty/

[8] http://lucene.apache.org/

[9] http://lucene.apache.org/solr/

[10] http://clerezza.apache.org/

JAX-RS and Oracle Jersey

JAX-RS is the java standard for building REST applications. The reference implementation is Oracle Jersey which is used within Apache Stanbol. Unfortunately Stanbol was using an older version not supporting the latest version of the standard. With the patches contributed to Apache Stanbol, we also contributed an update to the latest Jersey version, which allows a better compatibility with Clerezza.

Apache Sling

Apache Sling components are used for the automated deployment from the integration server (Jenkins) to a development instance of the platform. Like Stanbol, the Fusepool Platform is also available as an executable launcher created using Apache Sling.

Apache Jena TDB

Jena TDB[11] is used as triple stored backing Clerezza. It has been chosen because it offers better scalability than Sesame or Mulgara for which Clerezza also offers a backend. Implementing backends for Clerezza is relatively easy and backends for other stores are provided by third parties so if any problem with this backend occurs we could quite easily change.

Apache Stanbol

Stanbol resulted from the IKS FP7 research project as a software to bridge the gap between content management systems, powering the traditional Web and the linked data Web. While Stanbol offers a wide range of tools the most popular one is the so-called Enhancer, which is used together with the EntityHub to discover entities in unstructured documents. The Stanbol enhancer provides an extensible mechanism to process and semantically enhance contents. Stanbol comes with a set of components to be used in this enhancement chain, including components for transformation to text as well as for NLP analysis. The Fusepool Platform relies on these components for text processing. Stanbol was also the starting point for our Web-Application and security framework, which we contributed back to the Stanbol project.

Java Authentication and Authorization Service (JAAS)

JAAS implements a Java version of the standard Pluggable Authentication Module (PAM) framework. This security mechanism is supported by the standard Java Library as well as the Clerezza RDF framework. In this way, the security components Fusepool donated to Stanbol are based on this standard. In a nutshell this security framework allows assigning users and group permissions, such as reading a specific RDF graph.

2.3. Overview of contributions to above mentioned projects

As per now, the following are Fusepool developments that have been integrated in the upstream code base.

Apache Clerezza

- Creation of bundlelists for easy creation of launcher
- More flexible mechanism for smushing identical resources. Smushing allows to unify identical resources, i.e. to use one canonical URI in all triples about a resource.
- Update to use newer version of Lucene (for compatibility with Stanbol)
- TypeHandlerSpace implementation based on JAX-RS 1.1. TypeHandlerSpace is a Clerezza feature that allows associating JAX-RS resources to RDF types.

Apache Stanbol

- Introduction of authentication and authorization.
- Major refactoring of common libraries leading to the version 1.0.0 branch. Thanks to this refactoring it is much easier to use Stanbol components in other OSGi applications.

[11] http://jena.apache.org/documentation/tdb/

- New mechanism for UI creation based on rendering RDF resourcing using LDPath Template[12].
- User Manager for managing users, groups and permissions.
- Maven Archetypes for easy creation of various types of components.

2.4. Core Fusepool components

The major components of the Core Fusepool Platform are the Enhance Content Store (ECS) and Annostore. The deployed and downloadable platform also contains the following modules contributed by other work packages: Fisrtswim, dictionary annotator and the NER annotator.

Enhanced Content Store (ECS)

For storing unstructured documents and their metadata we initially planned to use the Stanbol ContentHub component. This component stores the document metadata in a Clerezza RDF store and separately maintains a Solr Index to allow functionality like faceted search. It turned out that this separation of the semantic information from the searchable one imposed severe limitations. For example it was not possible to retrieve the meta-data of an entity used as a facet as this was only available as a text in Solr without any link to the metadata in Clerezza. Another limitation was that by exposing directly the Solr Web endpoint it didn't allow for access control as Solr doesn't support this. As a replacement for this component we built our own Enhanced Content Store (ECS)[13]. In ECS the index is built using Lucene directly on the Clerezza graph. ECS exposes a semantic REST API for search, this is an API a client only needs to understand RDF and the used ontologies to fully use it, no further documentation is needed.

Annostore

The Annostore[14] is a component that allows storing and querying annotations. The annotations are described using the open annotation ontology[15]. The stored annotations can be accessed via the Annostore HTTP API or accessing the annotation graph via the Clerezza API.

Firstswim

Firstswim[16] is the JavaScript Based GUI client for the Enhanced Content Store. It allows full-text and drill-down search of the documents.

Fusepool NER

Fusepool NER[17] is an Apache Stanbol enhancement engine, it is based on the Stanford named-entity recognizer. Named-entity recognition or entity extraction is a process when special expressions that refer to unique real-world entities (like persons, locations, etc.) are extracted from the input text. NER works with predefined models from different domains.

Fusepool SMA

Fusepool SMA[18] is an Apache Stanbol enhancement engine, it is based on the Aho-Corasick string matching algorithm. It was created as an alternative solution for entity extraction, because it is much more flexible than any model-based NER solution. The Aho-Corasic string matching algorithm is a dictionary-matching algorithm that locates elements of a finite set of words and expressions which is called the dictionary, within the input text.

[12] http://marmotta.apache.org/ldpath/template.html

[13] https://github.com/fusepool/fusepool-ecs/

[14] https://github.com/fusepool/annostore

[15] http://www.w3.org/community/openannotation/

[16] https://github.com/fusepool/fusepool-firstswim

[17] https://github.com/fusepool/fusepool-ner

[18] https://github.com/fusepool/fusepool-sma

2.5.Execution/implementation

Since all of the partners involved in this work package were highly experienced in their respective fields and also in formal software development, a traditional approach for fleshing out the necessary specifications was used. This process proceeded as follows:

1. Partners discussed the exact requirements of the system. In this case we knew we needed an RDF storage and query platform. This platform needed to have the ability of ingesting and processing large amounts of information. Also, we knew that we needed to support full text search, machine learning components and user management. The main driving factors for this discussion were the individual deliverables required in WP3, WP4, and WP5. Since each of the partners involved in this WP were also involved in the other mentioned WPs, there was a tight sense of ownership and understanding of exactly the pieces required to make such a system.
2. Partners examined existing software, especially open source. As a European project, we were especially interested in using open source software to enable free-sharing of our final platform so that others could benefit from our work. Additionally there were no closed-source software which could meet our exact demands. As a result, the partners reviewed all available projects trying to find and take advantage of any synergies which existed between them. In the end, we discovered that the Apache Stanbol project met most of the needs of our final platform, and with a small amount of work, which is re-contributed to the open source community, we could bring it up to speed.
3. Partners sketched out a suitable platform around Stanbol . Through iterative discussions with the Stanbol community, we were able to determine the functionality which was missing from Stanbol. These included:
    a. Authorization: The Stanbol code correctly work with an active Java Security Manager. As a minimum this requires that all SecurityException are propagated. Ideally components clearly define what permissions they require and execute certain code blocks as privileged code.
    b. Authentication of users: for document security to work well, we need to be able to authenticate which user is asking to perform the actions
    c. User management: lastly, we need to be able to actually manage users, such as adding, deleting and assigning security levels.

    Additionally, we identified severe deficits with the UI and REST interface support in Stanbol so we contributed massive refactoring to this layer in Stanbol. We had a strong need for rapid and compliant development of new Stanbol components, so we produced Archetypes for the Apache Maven build tool which benefited our partners in later workpackages. One of these archetypes is for building Enhancements Engines. Other archetypes facilitate usage of the tools for providing UI and REST interfaces which we also contributed.

2.6. Solution

As a result, we were able to create the diagram in Figure 1 which we used as a framework for later tasks:

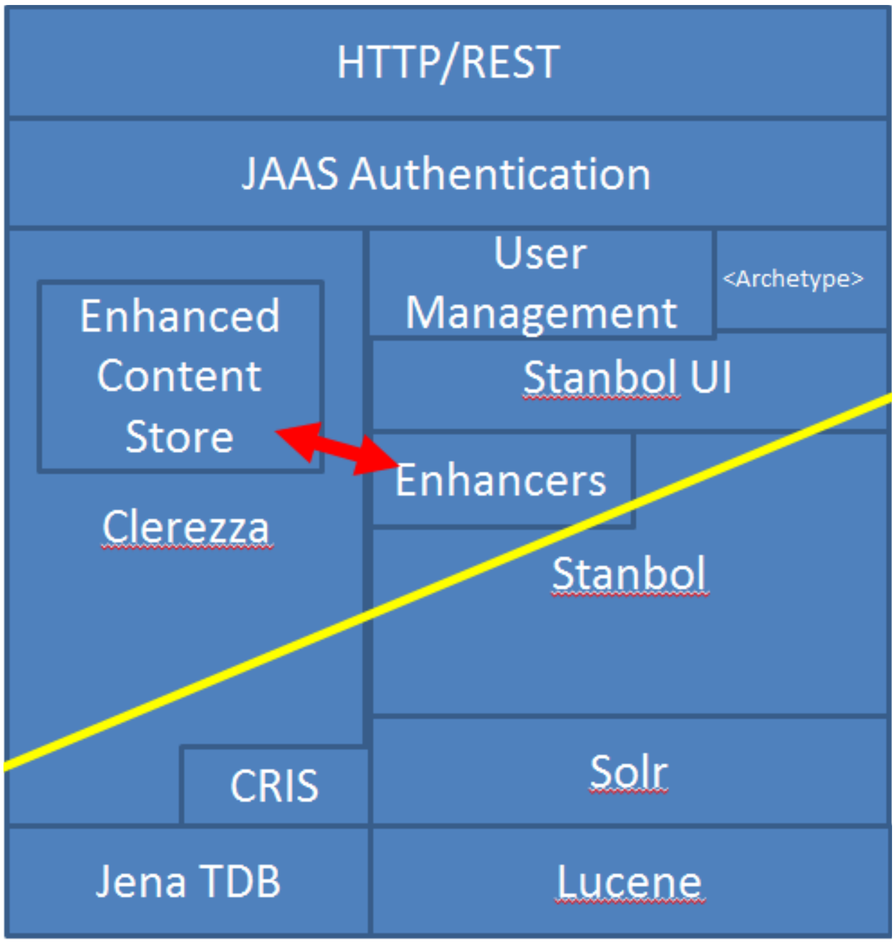


Figure 1 Fusepool platform architecture

As we can see, information is transmitted following REST patterns via HTTP. Then:

1. It goes through an authentication component which uses the HTTP user based authentication mechanism to verify user credentials and associated permissions based on a access rules described in an RDF graph. This graph describing users, roles and permissions is managed with the user manager. The User Manager is based on Stanbol UI Libraries and is one of many UI components (new ones can be built easily using archetypes). The yellow line in the diagram represents the authorization barrier. As any component can check for and require particular permission this is shown as a crosscut line though the modulesonly authenticated requests can cross this barrier ensuring that privileged information is protected.
2. Next we see that Stanbol uses enhancers to modify or perform additional actions on the information. These processes include detection of people, places, things, diseases, etc. Additionally they create links between the data in the incoming document and other documents allowing for a richer search experience. The Stanbol enhancers which are used to process the information were generated via the archetypes. The ability for new developers to easily extend Stanbol's power by using a generic archetype was one of our main contributions to the open source project.
3. These enhanced bits are stored in an Enhanced Content Store (ECS) which, like Stanbol, uses another Apache project, Clerezza, as its backend. Clerezza provides the RDF Storage (based on Jena TDB) as well as an indexing servicel based on Lucene, this allows for the rapid search of full-text and also excellent scalability.

4. Note that while our project does not directly use the Stanbol UI, as it is being extended and improved by BUAS, it still exists and gives a quick way for browsing through the information, a requirement of the CMS type persistence of the work-package.

Using this approach, we were able to re-use and integrate many existing open source projects. Adding some programming "glue" between them made their individually powerful properties extremely synergistic resulting in a fully empowered Fusepool platform.

Authentication and authorization in the Fusepool platform

When the fusepool project began Stanbol had virtually no security mechanism. The only security mechanism it contained was inherited from the underlying Apache Felix OSGi container which provided authentication for the Felix Webconsole. This authentication mechanism only allowed authentication of single user (the admin user) and only secured the OSGi management console. Users without the respective password could not perform OSGi administration task like installing or uninstalling OSGi bundles or shutting down the system or part thereof, conversely users with the password would have all privileges including the right to run arbitrary code. Apart from this restriction to the administrative interface Stanbol had no authentication or access control mechanism. Neither the components performing enhancement of text (a potentially costly operation requiring access to remote services) nor the storage of content in the contenthub or the creation of entities in the entityhub would require any authentication.

It is quite clear that this situation was not acceptable for the Fusepool Platform. The Fusepool Platform requires security mechanisms for the following reasons:

- System security: preventing unauthorized manipulation to the system
- Reduce the possibilities for denial of service attacks (DOS)
- Privacy of Data
- Restrict access to functions for economic or legal reasons
- Prevent the platform from being used as a relay for illegitimate access to third party systems and infrastructure
- Foundation for multi-tenant applications

A related issue not strictly pertinent to security is user customization. Users should be able to have their application preferences stored and the personalization applications supported both for UI as well as for machine learning applications.

An evaluated solution was to add security on the level of HTTP Proxy. By the architecture and stateless nature of HTTP it is possible to restrict access to certain resources using a so called *reverse proxy*. With a reverse proxy HTTP requests against the platform are first handled by the proxy before they are forwarded to the actual Stanbol based platform implementation. A reverse proxy can take care of authentication. It can associate permission requirements to specific resources and analyze HTTP requests to see if they contain matching credentials. If the request contains such credentials it is transparently forwarded to the actual platform server, if no such credentials are provided the proxy denies the request and challenges the client to provide authentication.

Without modification to Stanbol it would have been possible to implement such a reverse proxy based solution. However a closer examination showed that such an approach would have severe shortcomings:

- We need more fine grained access control, for example the SPARQL endpoint might be broadly accessible but only privileged user shall be allowed to execute SPARQL queries against graphs containing sensitive information.
- There would need to be a user management external to the platform. It wouldn't be possible for applications on the platform to provide additional functionalities specific to the current user.
- The access control rules based on HTTP resources require a high effort of configuration. Only slight changes to the HTTP API require reconfiguration of these rules.

- It is hardly compatible with the envisaged RESTful API: REST mandates the resources to be discoverable from the responses of previous requests (this is the so called hypermedia requirement of the REST architecture). Proxy based security by contrast requires a static set of HTTP resources.
- It doesn't leverage built in security mechanisms of the Java platform and used libraries.

As these shortcomings ruled out the option of having proxy based security we had to opt for an approach more tightly integrating security into the platform. The chosen approach leverages existing security mechanisms built into the Java Platform[19] and some used libraries like Apache Clerezza[20]. The java security model is based on a "sandbox" in which the code has limited and well defined access rights to resources both within and outside the Java Virtual Machine (JVM). When Java applications are run in secure mode at various points of their execution security checks might take place, execution only continues if the code is executed with sufficient permissions, otherwise a SecurityException is thrown. Places where such security checks take place include the standard libraries for accessing files or creating network connections. In the Apache Clerezza libraries which are used to access underlying triple stores such security checks take place before reading or writing to a graph.

The Permissions system is very fine grained so that different operations can all have dedicated permissions. However it is also possible to have more coarse grained high level permissions and then have code sections which are executed as privileged executed without permission check. These patterns allow system administrators to more easily manage the permissions of users. For example if a user has the permissions to send emails the administrator doesn't need to additionally give the user the right to access the network, the high level permission "Send Email" implies more fine grained permissions like "Open Network Socket".

Even though the actual security architecture is already integrated in Java, introducing such a system in Stanbol and the Fusepool platform required efforts at different levels:

- Social aspects in the Stanbol community
  - Raise awareness in the Stanbol community for the security related issues
  - Educate developers to write code in a way that runs in a secure environment
  - Advocate that problems arising after introduction of security are not a consequence of security but of the pre-existing broken code
- Technical aspects
  - Introduce a policy for user based security checks
  - Introduce an authenticating filter matching HTTP authentications to subjects in the Java permission systems
  - Improve Stanbol component to have sound permission requirements
  - User management
  - Replace the access control mechanism of the Felix Webconsole with the platform-wide system

The technically most challenging aspects were the introduction of the User Management and the authenticating filter.

Consistently with the design paradigms of the platform the users are described in RDF. This security sensitive information is stored in a system graph. The system graph has itself the most restrictive access control settings. In this system graph users are mapped to roles and roles and users are matched to permission. The permissions are described using the standard syntax to describe them as it is used in Java Policy files[21], note however that no policy file is used in the platform as such a file based configuration

[19] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html

[20] http://clerezza.apache.org/

[21] http://docs.oracle.com/javase/6/docs/technotes/guides/security/PolicyFiles.html

would be alien to the platform design. For describing the users and roles standard ontologies like FOAF and SIOC were used wherever possible.

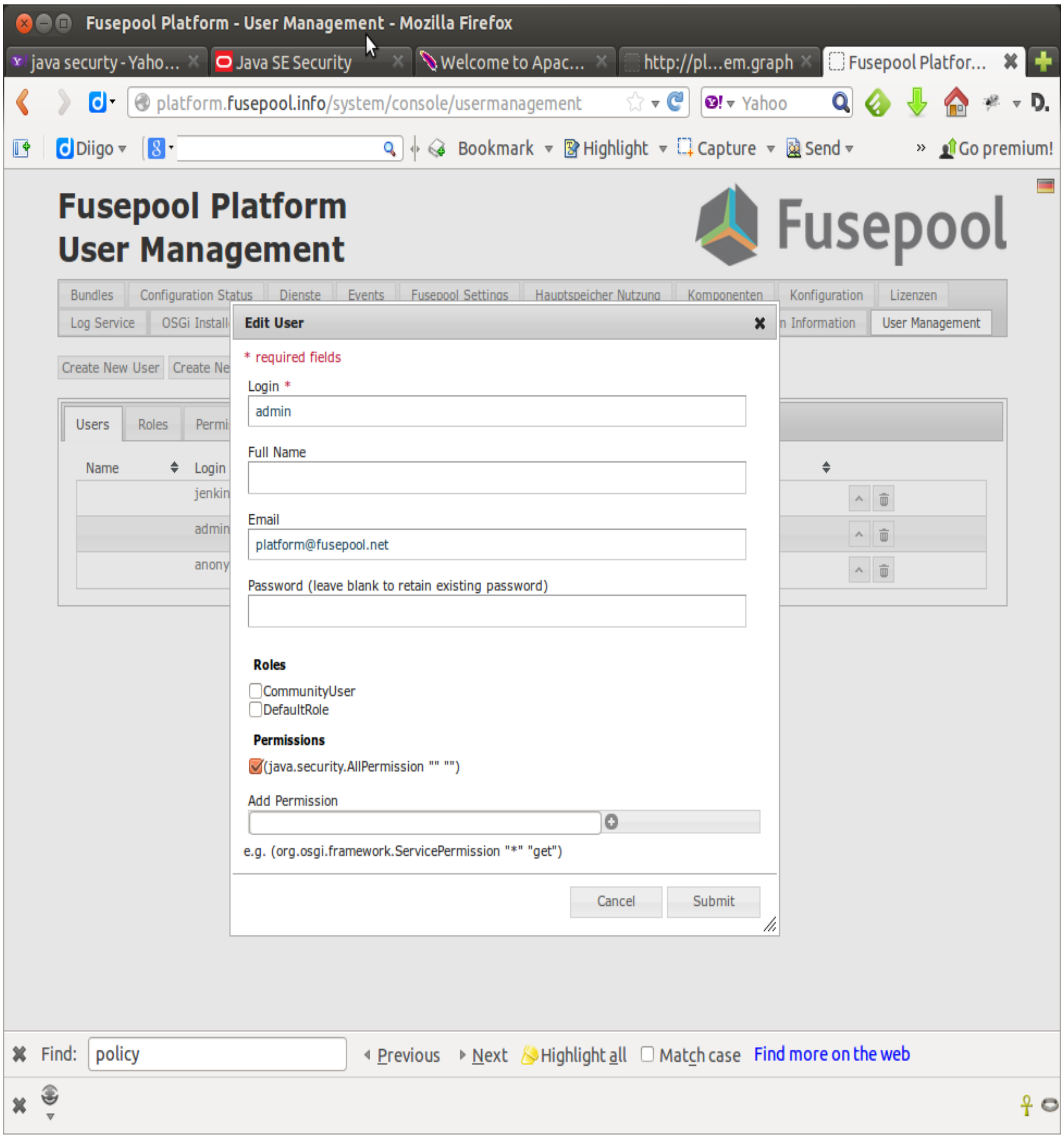
The following is an extract of the system graph describing the admin user:

```
<rdf:Description rdf:nodeID="A20">
        <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Agent"/>
        <zz:hasPermission df:nodeID="A21"/>
        <zz:userName>admin</j.2:userName>
        <foaf:mbox rdf:resource="mailto:platform@fusepool.net"/>
        <zz:passwordSha1>41c7fa920dc5462c33d5d0e3bba87d05e32</j.0:passwordSha1>
        </rdf:Description>
<rdf:Description rdf:nodeID="A21">
        <rdf:type rdf:resource="http://clerezza.org/2008/10/permission#Permission"/>
        <zz:javaPermissionEntry>(java.security.AllPermission "" "")</zz:javaPermissionEntry>
</rdf:Description>
```

Figure 2 Example system graph

The actual authentication is implemented by an HTTP Filter. This mechanism provided by the Java Servlet Standard somehow resembles an HTTP reverse proxy described above. However unlike the proxy a filter runs directly within the Java Virtual Machine. Also it only takes care authentication, not of authorization as a reverse proxy would have to. The authenticating filter analyzes the request headers and makes sure the request is subsequently executed with the privileges of the user that submitted the request. If no authentication credentials are provided in the request it is handled as a special “anonymous” user. If anywhere in the code executing the request a permission is needed to perform a particular operation it is checked if the executing user has this permission. If the user does not have the permission an AccessControlException is thrown. This exception will be caught by the authenticating filter which will send the client an HTTP 401 response. HTTP responses with this status code allow the user to authenticate and resubmit the request.

We also provided a graphical interface that allows management of the user and roles. This interface seamlessly integrates within the existing management console.



* required fields

Login *: admin

Full Name:

Email: platform@fusepool.net

Password (leave blank to retain existing password):

Roles

☐ CommunityUser
☐ DefaultRole

Permissions

☑ (java.security.AllPermission "" "")

Add Permission

e.g. (org.osgi.framework.ServicePermission "*" "get")

Find: policy

Figure 3 Illustration of the user interface to edit users from within the management console

All security related code was contributed to the Apache Stanbol project and accepted by the Stanbol community. It is now part of the Stanbol distribution. This is a very important aspect as only having the community develop new components in a secure environment ensures that they are developed taking the relevant security considerations into account.

2.7. Future work

At this juncture the expectations of the system fully match the specifications put forth in the work package. An important aspect for the Fusepool Platform which was missing in Apache Stanbol was security. Fusepool successfully contributed a solution leveraging the standard Java mechanism as well as RDF technologies. Both technical as well as social aspects had to be addressed for these security components to become part of Apache Stanbol. As stated in the DoW "this task requires defining the cloud computing platform for integration of components developed in work packages", which we have successfully done with great results. This is a pleasant surprise as it means that there is no current future work necessary or planned for this task.

There are a few additional places of future work. For example, it is well known that while the UI manager is fully functional, there are still some bugs which can only be addressed as they come up.

Currently the administrator is responsible for adding the users. While this is sufficient in a demonstration low-load environment, a long term goal is to add a simple user self-registration process so that users can automatically add themselves directly from the homepage allowing for a larger user base without unnecessary overhead on the administrators.

Additionally, Stanbol is single-tenancy. This means that there can be only a single platform (and thus target audience) targeted at any given moment. The obvious approach for a robust usage environment is the enabling of multi-tenancy which would allow many platforms to co-exist inside of the same java virtual machine, thus saving on resources and overall resource management. This is similar to Solr having multiple collections which are all acted upon and treated independently.

Lastly, another idea is the extension of the authentication to use OpenId. This is similar to the idea of facebook/google account logins which have become popular recently. This allows for a user to have a single sign on location, such as Google, and use their respective identity, via conformation and a swapping of tokens, on 3rd party sites without having to provide a password directly to them. This trend has become popular as there seems to be an overabundance of new services requiring new usernames and passwords. For example, Jira now supports Google authentication. It only makes sense that a professional level research tool meets or exceeds similar standards as proposed by other websites.

The currently most promising approach to authentication is WebId[22]. With this emerging standard the client authenticates by using a Client Certificate which bases on an asymmetric encryption algorithm. The server verifies the client certificate by retrieving a personal certificate of the user on the Linked Data Web. This way the user is authenticated by a profile of her choice and the server can use social distance metrics to assign permission. The technologies used in the Fusepool Platform match together very well with WebId, so implementing WebId would be an obvious extension of our linked data application.

[22] http://www.w3.org/community/webid/

3. Cloud platform and component implementation (Task T2.2)

In this task we created a suitable environment for both development and production of the platform. Using the development platform as an example, we were able to extrapolate and plan the associated production environment using currently available cloud resources.

3.1. Background

With T2.1 completed, an actual implementation of the schematic was needed. This implementation is not only a software problem, but also a hardware problem as the different components of software have different requirements. The goal then is to identify exactly the hardware needs required and how to best meet those cost effectively. Without a successful completion of this task, there is no production ready environment capable of delivering the platform to the users, so a well-designed and instantiated instance is of deepest necessity.

3.2. Execution/implementation

We experienced a slightly different approach to completing this task. While a cloud based system is the final deliverable, considering the long development time of the project, we decided it wasn't monetarily efficient to have a working production environment from day one. As a result, within the infrastructure of BUAS, a centralized development location was created so that the various partners and developers could have a single point of test and access. While this system has been running successfully for many months, it gave us a very practical experience as to the software requirements necessary for the system. Knowing the minimum software and hardware requirements for development leads to the very easy corollary of creating a suitable production environment. The necessary pieces in that respect are the adding of high availably, ensuring that there is no single point of failure.

Through a series of meetings, it became understood that all pieces of software required a Java operating system to run. Additionally, considering the large size of the data (for example, Pubmed alone is over 50GB when fully indexed and analysed) a large storage facility was needed. During peak testing time, over 20GB of ram is used for indexing and analysing the data quickly, though this requirement is much less during actual user-usage. This leads us to believe that a large amount of ram is necessary preliminarily and then during operations much less memory can be used.

Overall, there is a silver lining on this cloud. Since we have a sane development environment which workers can use to push their respective data for testing, we can reduce the overhead and cost of the cloud installation. This is done by using the development environment (which currently has over 60GB of ram), to do all of the respective indexing and updating, i.e. the most expensive processes, and then transfer over their persisted files to the cloud infrastructure allowing the front end machines to cheaply serve requests.

3.3. Solution

As mentioned above, our solution is the most cost effective way within which to proceed. By using a development environment to do the heavy lifting, the final cloud infrastructure need not be nearly as expensive due to high resource utilization.

Since the development environment will be used as the seed for the production environment, the migration plan which needs to be put in place consists simply of copying the data directories (Solr and Clerezza) directly to the same respective directories on the cloud. This process also allows for the creation of an off-cloud backup should anything happen which destroys the hosted information.

In order to provide high availability (HA), we take the standard approach of scaling horizontally using a load balancer. We can easily create this type of infrastructure using at minimum 3 amazon cloud machines, 1 m1.small and 2 m1.large in the following configuration:

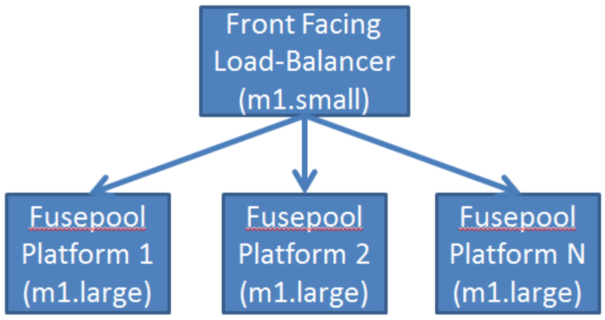


Figure 4 Cloud based infrastructure diagram

As we can see, there is room for additional horizontal growth by adding additional replicas alongside the main machine. The scalability here is best viewed by an example scenario. We can start with 2 instances during all times, to ensure that there is always a running backup should something unexpected go wrong with a single machine. The load balancer then splits requests between the machines to ensure that no single machine gets all of the traffic and becomes overloaded. The Amazon cloud load balancers support metric measuring which enables our approach to launch additional resources, and register them to the load balancer, automatically so that when the load balancer detects that machines are struggling it can automatically compensate. Additionally, it has the ability to shutdown machines should they become underworked, allowing for a savings in both energy and money for the platform.

Ongoing Platform Development

The need for ongoing development of the platform arises on one hand from the functional requirements and the experiences collected developing components. On the other hand it's also the ecosystem in which the platform lives that calls for continuous development.

To be is to be together with others.

A crucial factor determining the sustainability of the platform is its ability to interact with other systems and to benefit from the developments done elsewhere. Three distinct layers of interaction can be identified:

- Interoperability of software components
- Interoperability of data
- Exchange with relevant communities

All our development efforts in the field of the platform aim to ensure and improve interaction on one or multiple of these layers. So before going into more detail of the actually developed components it's worth looking into some more details of these layers of interaction and in how far they affected our activities.

Interoperability of software components

The Fusepool Platform is based on the OSGi architecture. Many projects use this architecture and provide components running within OSGi. Stanbol aims to provide a set of reusable components. The Fusepool Platform uses many of these Stanbol components alongside components from other open source projects such as Apache Clerezza, Apache Sling, Oracle Jersey.

Unfortunately just being conformant with the OSGi specification doesn't guarantee true interoperability. Besides adhering to OSGi componentization it is also crucial to expose APIs and require services and libraries in a way that no unnecessary coupling is tying components together. Especially with regards to the user interface it turned out that Stanbol is far from offering components that are easy to be reused. In fact despite being implemented against the JAX-RS standard[23] the Stanbol components have dependencies on

[23] https://jax-rs-spec.java.net/

proprietary interfaces of the JAX-RS implementation used in Stanbol. That way many Stanbol components cannot be used in another OSGi environment even if the other environment provides JAX-RS. Also many Stanbol components were using a proprietary method to access services. As a consequence of this the components only work together with the Stanbol container.

Another current limitation of most Stanbol components is that it is not possible to change the UI or to use the component without UI (but with a REST interface). For Fusepool it is however necessary to write detached front-end components as they are developed with WP5.

While OSGi and adherence to good development practices ensures interoperability of components running within the same Java Virtual Machine (JVM) for interaction between applications running in different environment the interaction is done via HTTP and ideally via interfaces designed following the REST principles[24]. The Frontend components we contributed to Stanbol foster the development of components that qualify as semantic and RESTful. This means that a client only needs a single service URI and understand the ontologies being able to fully use the provided services.

Interoperability of data

The vision of the semantic web is inherently decentralized. The resources which includes all the ontological terms are distributed. A key aspect is that the data uses dereferenceable URIs, that is identifier that can be dereferenced as to learn more about the identified resource. The web arising from this linking is referred to as Linked Data Web.

In the Fusepool Platform we have switched from the originally used Stanbol Contenthub to a new component[25] which among other advantages provides better integration in the context of Linked Data. The new system allows one to upload documents so that both the document and their generated metadata can be dereferenced at persistent HTTP URIs as well as the integration of data from remote sites. This data is indexed locally and a copy of the triples is stored to allow fast queries however as soon as the user access one of described resources they are pointed to the original location. Thanks to this design Fusepool is a service provider in the existing distributed linked data web and not a closed data silo.

Exchange with relevant communities

Open source software allows the creation of forks. In a fork development is continued separately from the original software. It is often tempting to fork software and have the full freedom to adapt the software to one's own vision. However, while some open source forks have yield to new successful projects they usually split the community and thus reduce the overall productivity. Fusepool depends on its open source components to be continuously improved and the long term success depends on the components to remain alive even after the end of the funding period. For this reason forking is for us only an option of last resort. Fusepool developers are active on the mailing lists of relevant upstream projects like Apache Stanbol and Apache Clerezza and try to convince the community of the necessity of improvements. Acceptance and influence in the community also requires support or the project that goes beyond the immediate requirements of the derived software. Fusepool developers have helped to contribute documentation and training material to Stanbol.

The Fusepool code that does not currently fit into any existing project is publicly accessible on GitHub[26]. There both components provided by project partners as well as the existing open source components are continuously integrated. We also use the GitHub bug tracking facilities to ensure transparency on open issues as well as to encourage third party users to report issues they might encounter.

[24] REpresentational State Transfer (REST) is an architectural style, defined by Dr. Roy Fielding's PhD thesis, *Architectural Styles and the Design of Network-based Software Architectures*. See http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[25] The Fusepool Enhanced Content Store is available here: https://github.com/fusepool/fusepool-ecs

[26] https://github.com/fusepool

3.4.Future work

As we can see, the architecture provided above meets the needs of the platform system across various scenarios, allowing it to both scale up and scale down accordingly. The implementation of such a system is very trivial and is thankfully well studied in industry as many other systems have very similar requirements to our own.

One possible place of improvement, which cannot be assessed currently due to the incomplete nature of the entire system, is the consideration of using Platform as a Service (PAAS) instead of Infrastructure as a Service (IAAS). Since all currently used components use Java, it should be possible to push the system into a remote Java virtual machine infrastructure such as Cloud Control. The basic advantage to such an approach is that it is much more fiscally efficient at often times a fraction of the cost of having an entire virtual machine. As mentioned, though, it is impossible to make this determination until the platform is completed and has all of the systems in place, which is not expected to happen until at least the 20 month milestone.

Another place of future work is in the contributed code and design patterns to Stanbol that resolved the various issues discussed above. Unfortunately this code and patterns are currently only used in a minority of the Stanbol components. We've agreed with the Stanbol community to create a stanbol-ng (new generation) branch containing only components satisfying some minimum portability requirements. We have already ported some components to this branch and hope that more members of the community will participate in this work. In the near future we will have ported all the components actually used within Fusepool so that we can start using this branch as long as the changes haven't become part of the main development trunk. We will also contribute to Apache Clerezza so that its components work well together with this improved Stanbol version. Fusepool already integrates components from both projects.

4. Data access and gathering with basic data curation (Task T2.3)

Once the system was in place, as per Task 2.2, we needed to move onto the process of importing the data into the system so that it could be used by the future work packages which create the user experience.

4.1. Background

This task defined the need for a data import policy from various data sources. Without the successful execution of this task, there would be no data in the system making it basically an empty search engine. A huge amount of data is available in semi-structured or structured format such as XML documents, CSV files, spreadsheets or tables in relational databases. In our case patents and PubMed publications were available as XML documents and FP7 calls as spreadsheets. These formats and their structural elements lack an explicit semantics. In order to make that data available as RDF a strategy is to map these elements to RDF resources and predicates defined in an ontology, that is a formal specification of a conceptualization. In ontology engineering this task is known in different communities, from developers to ontologists, as RDFize, data lifting or re-engineering of non-ontological resources.

4.2. Execution/implementation

The first step was to decide a strategy to import, store, index and transform the data. The dataset were chosen to support the case studies defined in D1.1. Three datasets where needed, one for each of the domains of interest: patents, articles from scientific journals and FP7 calls. Patents are available under different licenses from public and private organizations. Patents are related to many fields of research and a huge number of them come from biology and the life sciences so that PubMed was chosen as a source for scientific articles as it provides a huge collection of publications. The dataset for FP7 calls have been provided by Euresearch. The documents were to be stored and indexed within the Fusepool platform to make them available by keyword search and to extract semantic information from them through NLP processes and transformation in RDF.

The Fusepool platform provides a component, The Enhanced Content Store (ECS), based on Clerezza, that can store and index structured content and send this content to a chain of enhancers to process them. The enhancements extracted by each enhancer in the chain as RDF data are then stored in a graph that can be accessed and filtered using the Clerezza API. As the documents are semi-structured indexing them would include tags and structural elements in the index. In order to avoid the inclusion of structural elements in the index a plain text part is made up from selected properties values of the RDF triples like title, abstract and description. Such plain text part created from a document after it has been transformed in RDF is used by the ECS for indexing.

The transformation of the documents in RDF took a lot of effort as it is not just a transformation from one XML format to another but more a lift from a format without an explicit semantic to a semantically enhanced one where the meaning of each piece of data must be explicitly defined in an ontology. The procedure taken to map XML elements in the documents to terms in ontologies is described in the following paragraphs.

For each of the three datasets we had to take into account its size, format and standard. Patents and PubMed articles are huge datasets provided as XML files with different schemas. FP7 calls are provided as spreadsheets without a schema. Our approach in lifting all these datasets to semantic models was based on the NeOn methodology. We had to disclose the conceptualizations behind the datasets analysing the XML documents or spreadsheets themselves and other documentation available from the data providers' web sites. Some information has been given by email from EPO (European Patent Office) for patents and Euresearch for FP7 calls. We also got an understanding of the competency questions to which the ontology should provide answers looking at the EPO and PubMed websites where users can make searches following different criteria.

4.3. Patents Data Analysis

Patent offices guarantee people, companies and organizations the protection of their investments on research and of their intellectual properties. Patents confer the right to prevent third parties from making, using or selling an invention without their owners' consent. Patent data are used by companies to investigate potential competitors, or competing technologies to avoid investments on already acknowledged patents or to find business partners. An inventor or company that has developed a product can request a patent submitting an application to a patent office. The office releases a grant if the patent doesn't infringe any other right.

Every industrialized country has a patent office. The European Union has its own patent office with a web site from which patent applications and grants can be accessed online, downloaded or requested. Users that access the EPO's web site can search patents by keywords or more advanced search by classification, inventors, publication date and other attributes. Users can send feedback for errors they might find while looking at a patent document.

Firms and legal studios perform much more sophisticated searches on patents on behalf of their clients. These companies need to download the data and analyse it using different technologies and approaches. We can consider different ways in which one can leverage the data made available by EPO or other patent offices. A patent document has metadata attributes, as said before, like the inventors' names, the status of the patent application, date of publication, country in which the invention has been made, claims and also a description of the invention in which the applicants must describe in which way their invention is new and different from other already patented inventions. From text mining the patent descriptions different type of automatic classification can be done that can be added to the categories provided by the patent office at the time the patent application was submitted. Patents can be also grouped in clusters based on the similarity of attributes values and sorted in order to facilitate the analyst in collecting all the relevant documents on a topic of interest.

Today there are four major classification systems for patents. A classification is a hierarchical organization of subject matter, often referred to as a taxonomy that helps to organize documents. The European Union uses the ECLA classification system, an extension of the IPC classification system used by many of its member states and other countries worldwide. The US uses the USPC and Japan uses F-term. These taxonomies are quite huge having around 150000 concepts. Patent documents can be downloaded from EPO in the EBD/XML format. The USPC uses a different, XML based, document format.

Some collections of patents are available for free. MAREC is a corpus of over 19 million patent applications and grants used mainly for research purposes. USPTO data sets are available from Google.

PubMed Data Analysis

PubMed is an archive of biomedical and life sciences articles published by different journals. The archive is maintained by the U.S. National Institutes of Health's National Library of Medicine. The articles are freely accessible from the PubMed website according its policy. A subset (Open Access Subset) of the archive is made available under Creative Commons or similar licenses. This subset contains articles in XML format that can be downloaded from an FTP service. A free license to use this dataset was requested and approved. The dataset is divided in folders each containing articles from a journal. The XML document contains bibliographic information such as title, abstract, authors and affiliation, license, citations, classifications and identifiers (DOI - Digital Object Identifiers).

FP7 Calls Analysis

The European Commission funds research and innovation programs through calls under the FP7 programme to which companies and organizations can participate. A call responds to challenges established by the EU Commission and targets one or more objectives. A company or organization must search in the EC portal among these objectives to which call it wants to participate. Each call has a title or number, description, publication date and deadline, area, funding scheme, budget allocated and topics. The dataset has been provided as a schema free spreadsheet by EUResearch, a Swiss organization that supports companies and institutions that want to apply to the EC FP7 calls.

4.4. Mapping the data to RDF

The transformation from semi-structured content to semantically enabled chunks of knowledge requires different steps. The first one is the selection of an ontology that provides concepts and properties that cover the domain of interest. The second step deals with the definition of patterns to create dereferenceable URIs for entities. Some patterns have been published and are well accepted within the Data community. The third step requires the selection of a mapping language to map elements from non-ontological resources to terms in an ontology. The result of this task is a schema alignment. If an ontology is not available or does not cover the domain in the required details, it has to be developed from scratch or extended. For the patent domain a very specific ontology was developed in an FP6 project called Patent Expert that provided terms to which all the elements and attributes in the XML document could be mapped. One issue with this ontology is that it is not maintained anymore so its terms cannot be dereferenced. PubMed documents are well covered by well-known ontologies such as Dublin Core and Bibliographic Ontology. A light ontology has been developed to deal with FP7 calls. As the format of the patent and PubMed documents is XML the XSLT language was chosen to define the mapping between these documents and their RDF/XML representation. FP7 calls were provided as non-standard CSV files, that is tabular data with only some hundreds of records. The approach taken to deal with this type of dataset is to transform its structural elements into RDF and then use SPARQL CONSTRUCT as a rule language to map the intermediate RDF to terms in an ontology to have the final RDF data. More details about the mapping are given afterwards.

4.5.Solution

Three RDFizers have been developed as Stanbol enhancers to transform documents from XML or CSV to RDF. The work has been organized in two subtasks. One subtask, Data Mapping, is devoted to the development of the transformations for all the documents that were chosen as source: patents from the MAREC corpus, PubMed articles and FP7 calls. The following subtask, RDFizers, was devoted at including such transformations into OSGi bundles in order to provide an easy usage of those transformations within the platform. The first two transformations were from XML to RDF/XML and have been developed as XSLT style sheets. The FP7 calls, originally provided as CSV files, have been transformed into RDF using SPARQL rules. Each subtask is described in the following paragraphs.

4.6. Data Mapping

Given the use-cases, data for patents, publications, was acquired using the wget command-line tool, and the funding calls and topics data was acquired over email. Patents consist of MAREC (MAtrixware REsearch Collection) which was retrieved from IRF (Information Retrieval Facility) [1] website containing nearly 20 million documents. Publications were retrieved from PubMed Central [2] website consisting of nearly 2.7 million documents. Funding calls and topics was acquired from EUResearch [3] privately consisting less than thousand documents for one year.

Vocabularies and Ontologies

Besides the common vocabularies: RDF RDFS, XSD, OWL, XSD, the DC Terms (DCMI Metadata Terms) is used for general purpose resource publications, and the FOAF (Friend of a Friend) vocabulary is used to primarily represent Agents, Persons and Organizations. PROV-O is used for provenance coverage. SKOS (Simple Knowledge Organization System) and XKOS (extension to SKOS) to cover concepts, concept schemes and their relationships to one another.

More specifically for the data at hand, the PATExpert ontology is used for patents data, BIBO (Bibliographic Ontology Specification) for main concepts and properties for describing publications, journals, bibliographic references. The team found it necessary to create a minimal vocabulary under the Fusepool namespace for funding Calls and Topics, as there were no suitable or dereferencable vocabulary or ontology currently in the wild for the data acquired from EUResearch.

Vocabulary mapping

The vocabulary that's created for EUResearch data was mapped to FP7-PP vocabulary which was created in EU project LATC (Linked Open Data Around The Clock).

Data cleansing

By in large, original data was used "as is" with the exception of some normalization e.g., trimming whitespaces or replacing spaces with dashes in the case of IRIs where applicable.

IRI design patterns

An outline for the IRI patterns that's used for the data is provided in table below. *authority* generally refers to fusepool.info in the case of data that's published under the Fusepool namespace. Entity types is a classification of the type of *things* which occurred in the datasets. / is used to tokenize terms, and some of the identifiers were normalized to be safe URIs. The design approach in general follows some of the best practices.

Entity type	**IRI pattern**
rdf:Class, owl:Class	http://{authority}/class/{id}
rdf:Property	http://{authority}/property/{id}
pmo:PatentPublication, bibo:Document, fusepool:Call	http://{authority}/doc/{id}/{sub-id} or http://{authority}/doc/{UUID}
foaf:Agent, foaf:Person, foaf:Organization, schema:PostalAddress	http://{authority}/id/{UUID}
skos:ConceptScheme	http://{authority}/concept/{id}
skos:Concept	http://{authority}/concept/{id}/{sub-id} or http://{authority}/concept/{id}/{UUID}
sdmx:CodeList	http://{authority}/code/{id}
sdmx:Concept	http://{authority}/code/{id}/{sub-id}
Context URI / Named Graph	http://{authority}/graph/meta or http://{authority}/graph/{id} or {UUID}
prov:Activity	http://{authority}/prov/activity/{UUID}

In cases where the identifiers that could be easily algorithmically constructed from the data source or follow a common pattern, they were used directly in the pattern in order to create predictable (resource-friendly) and human-friendly IRIs; unique identifiers which were available directly in the documents e.g., patent identifiers, patent classifications, publication identifiers, funding call and topic identifiers. In other cases where the occurrences of *things or concepts* cannot be absolutely differentiated from one another, UUID values were generated and used in the IRI pattern. The split between id and sub-id is intended to hierarchically separate some broad scheme or dataset (id) from a narrower concept or document (sub-id). We differentiate between concept schemes and codelists, and concepts and codes in a way that the codelists and codes are concepts that are typically codified, and may be part of well-known standards e.g., ISO 3166 country codes.

The decision for when to use machine-generated identifiers and when not was based on their best applicability. The acquisition of identifiers from documents which were essentially non-ambiguous were used for simple recall. In the latter case (machine-generated unique identifiers), since sufficient amount of context is always required in order to distinguish one resource from another e.g., the subject keyword "being" in multiple publications, or contributor named "Thomas A. Anderson", for practical purposes it is nearly impossible to absolutely link them given varying data quality across heterogeneous datasets. Corollary, these type of separations between resources is intended to be tackled in the reconciliation and interlinking steps, which are considered to be far more preferable than at time of data mapping and transformation. The ease of constructing the Fusepool IRIs for selected patterns also promotes discoverability and reconstruction by others - an invitation for incoming interlinks to Fusepool resources. See also the Enrichment section.

The paths in the IRIs are prefixed e.g., /property, /class, /graph, /prov, /doc, /id, /concept, /code, in a fashion that's fairly common in published Linked Data, unambiguous for practical purposes, and considered to be "simple enough" by the authors.

Enrichment

Some fixed interlinking at the time of mapping was done for publications. It included *same as* (owl:sameAs), *see also* (rdfs:seeAlso), and *is format of* (dcterms:isFormatOf) relations to existing, dereferenceable URIs on the Web i.e., under dx.doi.org, ncbi.nlm.nih.gov, biotea.idiginfo.org, linkedlifedata.com, identifiers.org, bio2rdf.org, europepmc.org, and hubmed.org . The rationale for this was due to interlinking to external datasets as accurately and efficiently as possible provided that the IRI patterns of the external resources contained the document identifiers in their paths. The efficiency here is in contrast to deriving the same relations in the reconciliation and the interlinking phases.

Additional labels were created to promote general purpose use and simplified application development. In patents data, rdfs:label was added in addition to foaf:name, foaf:firstName, and foaf:lastName for people and organizations. In the case of publications, foaf:name was constructed by joining foaf:firstName and foaf:lastName separated by space.

Tests: transformations and optimisations

As the patents and publications data was in XML, they were transformed to RDF/XML using XSLT 2.0. Saxon's command-line XSLT and XQuery Processor [23] tool was used for the transformations and generally ran as part of Bash scripts. For testing purposes, approximately 20000 patents, 1600 publications, and 3 funding documents were transformed, taking 16 hours, 21 hours, and within a minute respectively.

"Out of the box" quality of data mapping was checked in two ways. First, Jena Fuseki server was setup to provide sample data - approximately 11 million triples containing patents, publications, funding calls, and classifications - over a SPARQL endpoint with Linked Data Pages to serve requested resources. Using the follow-your-nose exploration pattern, authors navigated the resources in their HTML representation in a Web browser to get an intuitive feeling of their linkage and ease of discovery. Second approach was to come up with a set of natural queries which could be inquired about the data, then they were formulated with SPARQL and tested over the endpoint. Complexity of the SPARQL queries i.e., ease of formulation, as well as a brief check of their response times was the determining factor as opposed to rigorous benchmarking.

The takeaways for the patents data was that lessening verbosity to represent a patent document as well as its linkage to another patent document was necessary in order to optimize the discovery of related resources. For instance, in the initial design, to arrive at patent's list of contributors, or the filing office in which the patent was submitted to was unnecessarily too many nodes away. This was done sufficiently without stepping away from the general representation of a patent as described in the PATExpert ontology.

4.7. RDFizers

RDFizers for patents, PubMed articles and FP7 calls have been implemented as Java OSGi bundles that share a common interface to make them behave in the same way within a chain even if their job and results will be different. Each RDFizer receive a document and before processing it checks its MIME type. Documents can be sent via Java API or REST API. In both cases the content item MIME type must be application/xml for Patents and PubMed articles and text/csv for FP7 calls. The RDFizer try to process the content item. If the content item validates with the transformation the result RDF data is added to its metadata. The RDFizers add also a plain text part to their content item with MIME type text/plain. This part contains text taken from RDF properties that are relevant for indexing and information extraction by NLP enhancers such as title and abstract for patents and articles. This should improve the quality of indexing and information extraction compared to the use of the full content with its structural elements. The RDFizers can be installed into the platform from the console and then used within a chain. Stanbol uses a convention to define the REST API available when a new chain is instantiated and populated with enhancers. For example one can use curl to send an HTTP POST command

curl –u username:password -i -X POST -H "Accept: text/turtle" -H "Content-Type: application/xml" -T ep-1000000-a1.xml "http://platform.fusepool.info/enhancer/chain/patent_chain"

sends the file ep-1000000-a1.xml to a chain that has been configured with the name patent_chain in the Fusepool platform and expects a RDF/TURTLE stream as a result. The documents can be sent to the Enhanced Content Store for storage, indexing and enhancement via REST API using curl to send an HTTP POST message to the ECS

curl –X POST –u username:password –H "Content-Type: application/xml" –d @patent1.xml http://platform.fusepool.info/ecs

These commands are used within bash scripts to bulk enhance and upload documents.

4.8.Future work

RDFizers for XML to RDF transformations are similar in many ways and it should be possible to just point to the XSLT style sheets from one instance of the same bundle configured in the platform console. The same could be done with the transformation based on the SPARQL CONSTRUCT. A library based on HTTP URIs and protocol could be added as a layer in order to simplify the use of the datasets by software developers who do not know RDF or SPARQL.

4.9. References

[1] http://www.ir-facility.org/
[2] http://www.ncbi.nlm.nih.gov/pmc/
[3] http://cordis.europa.eu/ist/kct/patexpert_synopsis.htm

5. Outlook

It is a pleasure to say that the above mentioned tasks have been completed in an organized and well managed fashion. While there are always small hiccups due to various technological problems (version conflicts, bugs, etc), with a clear directive and excellent project planning, it becomes possible to solve these difficulties in a reasonable time period and thus meet project deadlines. As discussed above, the tasks in this particular deliverable were completed on time and according to specification. The interesting part is of course seeing how the various tasks fit together and how the co-operation of all partners leads to a successful outcome. Consider that the first task was planning a software architecture system. This task basically defined all of the problems and limitations that would occur for the rest of the project, but through good communication and various expert experiences, excellent decisions were made allowing for a seamless plan. The second task was to implement such a plan in a development and production environment. Again, due to the management of the first task, the second task became significantly easier to develop and put in place. Lastly, once the system was in place, ingesting the information became needed and was also mandated by the next task. With all of the above tasks in good working order, the next tasks for the platform are expected to go smoothly. This is of a critical nature as the next steps involve more complicated software and algorithms which will be what gives Fusepool its unique sales proposition.

6. Appendix A – Platform Installation Instructions

Fusepool-platform

The fusepool platform containing all fusepool developed components as submodules. As soon as this repository gets a commit the platform is built through Jenkins.

To get the source do the following:

```
git clone git@github.com:fusepool/fusepool-platform.git
cd fusepool-platform
git submodule init
git submodule update
```

To compile the fusepool platform and its modules you need to have Maven version 3 or newer installed.

On many systems to should set an environment variable for maven to be executed with enough memory:

```
export MAVEN_OPTS="-Xmx1024M -XX:MaxPermSize=128M"
```

Compile with

```
mvn install -Dmaven.test.skip=true
```

Of course, omit -Dmaven.test.skip=true if you want to run the tests as well

To run it change to launcher/target

```
cd launcher/target
```

and run it with

```
java -Xmx1024M -XX:MaxPermSize=400M -Xss512k -jar launcher-0.1-SNAPSHOT.jar
```

To start it in debug mode so that you can connect a debuger on port 8888

```
java      -Xmx1024M      -XX:MaxPermSize=400M      -Xss512k      -Xdebug      -Xnoagent      -
Xrunjdwp:transport=dt_socket,address=8888,server=y,suspend=n -jar launcher-0.1-SNAPSHOT.jar
```

Adding new Submodules

A few notes on adding new modules to the platform:

The submodule should be added by their https (not git-uri so that people without edit right can still check out the platform) e.g. git submodule add https://github.com/fusepool/fusepool-something.git

Add the submodule to the reactor, i.e. to the pom.xml in fusepool-platform

Add the module's groupId and artifactId to the bundlelist: bundlelist/src/main/bundles/list.xml