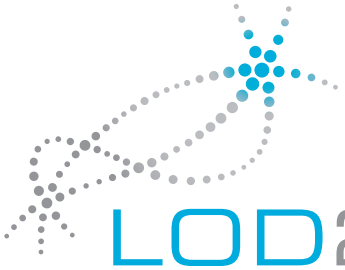

Collaborative Project

LOD2 - Creating Knowledge out of Interlinked Data

Project Number: 257943 **Start Date of Project:** 01/09/2010 **Duration:** 48 months

Deliverable 9a.2.1

Prototype of Matchmaking Web Services for Linked Commerce Data in the Domain of PSC

Dissemination Level	Public
Due Date of Deliverable	Month 34, 2013-06-30
Actual Submission Date	Month 36, 2013-08-03
Work Package	WP 9a, LOD2 for a Distributed Marketplace for Public Sector Contracts
Task	T9a.2
Type	Prototype
Approval Status	Approved
Version	1.0
Number of Pages	14
Filename	deliverable-9a.2.1.pdf

Abstract:
The deliverable describes a prototype implementation of web services for matching public procurement notices to business entities that might be relevant as suppliers to contracts advertised in the notices. The services' API is documented by describing its interaction behaviour and input and output data.

Project funded by the European Commission within the Seventh Framework Programme (2007 - 2013)

History

Version	Date	Reason	Revised by
0.1	2013-07-11	Initial version	Jindřich Mynarz
0.2	2013-07-14	Implementation	Jakub Klímek
0.9	2013-07-17	Version for internal peer review	all authors
0.9	2013-07-30	Internal peer review	Sebastian Tramp
1.0	2013-08-03	Version after internal peer review, added JSON-LD context	Jindřich Mynarz

Author List

Organization	Name	Contact Information
UEP	Matěj Snoha	matej@snoha.info
UEP	Jakub Klímek	klimek@ksi.mff.cuni.cz
UEP	Jindřich Mynarz	jindrich.mynarz@vse.cz

Executive Summary

We implemented a prototype of web services for matchmaking public procurement notices with business entities that might be relevant as suppliers to contracts advertised in the notices. In total, 3 matchmaking web services were developed, each of which may be subscribed to via a dedicated subscription service. In this way, users may configure the web services to receive alerts about business opportunities in public procurement, that are of interest to the companies they represent. Representatives of contracting authorities can vice versa use the web services to be notified of suitable suppliers to contract notices they issue.

The resulting prototype is implemented as a suite of Java Servlets. Documentation of the API exposed by them, containing descriptions of valid input and output, forms the main part of this text. To enable exploring its functionality a publicly accessible demo instance was setup. The instructions for deploying the software are included in the final section of the deliverable.

Table of Contents

List of Tables

1 Introduction

The deliverable reports on a prototype implementation of web services for matchmaking public contracts and business entities drawing on their structured semantic descriptions available as linked data in RDF. The matchmaking services exploit the work conducted previously in the LOD2 project, operating on data modelled using the patterns introduced in [1]. The functionality exposed by the matchmaking services can be integrated into other applications. It will be surfaced in the previously documented public contracts filing application [2] as a part of the work of an upcoming LOD2 deliverable D9a.2.2.

In terms of technologies, the web services are implemented as a Servlet 3.0[1] Java 7 project running on Apache Tomcat 7 and using Apache Jena and Apache Jena Fuseki triplestore. The compiled WAR file of the web services is available for download at `http://lod2.vse.cz/public/files/matchmaker/TenderStats-MatchmakerWS.war`.

Now we will address some of the implementation differences as compared with the LOD2 Description of Work. While most of the fundamentals of planned functionality was implemented, the prototype lacks matchmaking based on machine learning. It was not feasible to experiment with incorporating machine learning due to inavailability of sufficient training data. The public procurement datasets that we harvest contain pairs of contracts and suppliers, however, data about the business entities that responded to the contract's call for tenders and lost are not typically published. Therefore, only the "positive" examples for machine learning are available, lacking the "negative" examples of unsuccessful bidders. As a result, we decided to postpone reporting on the development of matchmaking based on machine learning to the subsequent deliverable describing the next iteration of matchmaking web services integrated into the previously developed public contracts filing application. Due to recent changes in legislation of the Czech Republic, some sources of public procurement data now publish unsuccessful tenders as well as those that were awarded the contract. We plan to harvest these new data sources in order to obtain data test the viability of machine learning for the purposes of matchmaking.

Due to simplicity of the initial prototype of web services their results are output in JSON instead of RDF. However, the services provide partial mapping of the output format to RDF via JSON-LD context. As recommended in the JSON-LD specification,[2] when the services are requested with HTTP Accept header containing the `application/ld+json` MIME type, the response is enriched with HTTP Link header pointing to a JSON-LD context that allows to interpret parts of the response as RDF. Other output formats, including RDF, will be considered for a subsequent version of matchmaking web services, based on requirements for integrating the results in other applications.

The final alteration made in the implementation with regards to the LOD2 Description of Work concerns the notification mechanism. Instead of using feeds as notification protocols as originally planned the prototype adopts email subscriptions to notify its users of relevant results. The principal rationale behind this decision was the recognition of supplier's information needs as confidential. Suppliers seek information that is relevent to their commercial interests, which they likely want to keep private. On the contrary, feeds are typically used and perceived as public resources, even though securing access to them using e.g., HTTP/1.1 authentication is feasible. Therefore, we decided to use email, which is a better suited medium for this purpose, and it is more common among the users than feed readers.

The resulting implementation thus, for the most part, corresponds to the requirements planned in the LOD2 Description of Work. In the next chapter, we document the ways in which users can interact with the matchmaking web services.

[1] `http://tomcat.apache.org/tomcat-7.0-doc/servletapi/`
[2] `http://www.w3.org/TR/json-ld/#interpreting-json-as-json-ld`

2 Documentation

The Matchmaker implementation consists of 3 web services. Each of them supports subscription to receive daily results. An example instance of the web services is hosted at:
http://xrg15.projekty.ms.mff.cuni.cz/ws/Matchmaker

Input parameters can be sent using `GET` or `POST` HTTP method. The application expects all non-ASCII text inputs to be encoded in UTF-8 encoding. The output format is JSON in UTF-8 encoding. GZIP output compression will be used if negotiated using HTTP. There is a simple web interface included for testing purposes available at:
http://xrg15.projekty.ms.mff.cuni.cz/ws/matchmaker/

In the following sections, we will describe the web services for getting similar contracts, suitable suppliers and suitable open calls for tenders. The concluding part walks through the steps for deploying the web services.

2.1 Similar contracts

Given a contract, the service returns list of similar contracts including contract URI and its details.

2.1.1 Input

The parameters required or accepted by the web service are described in Table 1.

Action:
http://xrg15.projekty.ms.mff.cuni.cz/ws/Matchmaker?action=getSimilarContracts

Minimal example:
http://xrg15.projekty.ms.mff.cuni.cz/ws/Matchmaker?action=getSimilarContracts&contractURI=http://linked.opendata.cz/resource/ted.europa.eu/public-contract/dd088300-0417-4b6a-bc2e-6485a6024434-176328-2012

Parameter	Required	Default	Possible values	Description
contractURI	use either this or contractRDF		URI	URI of an instance of the `pc:Contract` class
contractRDF	use either this or contractURI		RDF/XML, N-triples, Turtle	RDF serialization of the input `pc:Contract` instance
offset	no	0	integer	0-indexed offset of the first result to return
limit	no	10	integer or `all`	maximum number of results to return

Table 1: Similar contracts web service parameters

2.1.2 Output

The output is a list of similar contracts encoded in JSON. For each contract, values described in Table 2 are returned.

Field	Datatype	Description
contractURI	URI	URI of the current contract
title	string	title of the current contract
description	string	description of the current contract
price	string	price of the current contract
currency	string	currency of the current contract
place	string	place of the current contract
percent	float (percentage)	similarity of the current contract to the input one
comparerMessages	list of records	explanations of scores given by individual comparers (Name, Weight, Score, Details)
country	string	country of the current contract
distance	integer	distance in kilometers of place of the current contract from the place of the input contract
publicationDate	date	date of publication of the current contract
mainCPV	CPV code (string)	main CPV code of the current contract

Table 2: Similar contracts web service output description

2.1.3 Workflow

Here we briefly describe how the matchmaking process is executed:

1. The input contract is downloaded in RDF. Either we get the RDF represenation according to the Public Contracts Ontology[3] directly or we get a URI and look for it in the pre-configured local triplestore. If it is not found, we access the URI and request RDF data using HTTP content negotiation.
2. In the first phase we filter all contracts in our triplestore based on the similarity of CPV codes. The similarity here is the length of the longest common prefix of two CPV codes.
3. In the second phase we refine these results by applying additional comparers. These compare e.g.,
   (a) tender deadlines - the shorter the interval between the two tender deadlines, the bigger similarity
   (b) publication dates - the shorter the interval between the two public contract publication dates, the bigger similarity

[3] https://code.google.com/p/public-contracts-ontology/

(c) geographical distance - here we measure the distance between the places where the public contracts were executed. We use the Gisgraphy[4] framework for conversion of addresses to geocoordinates and *Haversine* formula and *Vincenty's* formulae for distance computation.

(d) textual similarity - we compare titles of contracts using the SoftTFIDF[5] algorithm. The problem here is that textual similarity is not applicable for titles in various languages.

4. A final score is computed for each contract and the list gets sorted and returned. Here we assign a weight to each of the above mentioned comparers and get the result as a sum.

2.2 Suitable suppliers

Given a contract, the service returns a list of URIs of suitable suppliers (`gr:BusinessEntity`) including their descriptions.

2.2.1 Input

The parameters required or accepted by the web service are described in Table 3.

Action:
`http://xrg15.projekty.ms.mff.cuni.cz/ws/Matchmaker?action=getSuitableSuppliers`

Minimal example:
`http://xrg15.projekty.ms.mff.cuni.cz/ws/Matchmaker?action=getSuitableSuppliers&contractURI=http://linked.opendata.cz/resource/ted.europa.eu/public-contract/dd088300-0417-4b6a-bc2e-6485a6024434-176328-2012`

2.2.2 Output

The output is a list of suitable suppliers encoded in JSON. For each supplier (`gr:BusinessEntity`), values described in Table 4 are returned.

2.2.3 Workflow

Here we briefly describe how the matchmaking process is executed:

1. The input contract is downloaded in RDF. Either we get the RDF represenation according to the Public Contracts Ontology directly or we get a URI and look for it in the pre-configured local triplestore. If it is not found, we access the URI and request RDF data using HTTP content negotiation.
2. A list of similar contracts is retrieved (see subsection 2.1).
3. For each similar contract its supplier is extracted to a list.
4. A similarity score is computed for each supplier based on its number of contracts, their total price and the number of contracting authorities making deals with this supplier.
5. The list gets sorted and returned.

[4] http://www.gisgraphy.com/
[5] http://dc-pubs.dbs.uni-leipzig.de/files/Cohen2003Acomparisonofstringdistance.pdf

Parameter	Required	Default	Possible values	Description
contractURI	use either this or contractRDF		URI	URI of an instance of the `pc:Contract` class
contractRDF	use either this or contractURI		RDF/XML, N-triples, Turtle	RDF serialization of the input `pc:Contract` instance
offset	no	0	integer	0-indexed offset of the first result to return
limit	no	10	integer or `all`	maximum number of results to return
sortField	no	score	`score`, `name`, `place`, `contractingAuthorities`, `progression`, `totalContractValue`, `contracts`, `avgContractSize`	sort results (before applying offset and limit) according to the specified field
sortOrder	no	desc	`asc` or `desc`	sort order: `asc` (ascending) or `desc` (descending)

Table 3: Suitable suppliers web service parameters

2.3 Suitable open calls for tenders

Given a list of CPV codes, the service returns a list of URIs of suitable open calls for tenders including their details.

2.3.1 Input

The parameters required or accepted by the web service are described in Table 5.

Action:
http://xrg15.projekty.ms.mff.cuni.cz/ws/Matchmaker?action=getSuitableOpenCalls

Minimal example:
http://xrg15.projekty.ms.mff.cuni.cz/ws/Matchmaker?action=getSuitableOpenCalls&cpv=30213000&expired=true

2.3.2 Output

The output is a list of suitable calls for tenders encoded in JSON. For each call for tenders (`pc:Contract`), values described in Table 6 are returned.

2.3.3 Workflow

Here we briefly describe how the matchmaking process is executed:

- We create an empty virtual contract that is described solely by the CPV codes given to the web service.
- We search for similar contracts to this one as in subsection 2.1.

2.4 Subscription service

There is a special subscription service that enables users to register their email address, their chosen web service and its parameters in order to receive the web service's results continually.

Web service address:
`http://xrg15.projekty.ms.mff.cuni.cz/ws/Subscriptions`

Minimal example:
`http://xrg15.projekty.ms.mff.cuni.cz/ws/Subscriptions?action=getSuitableOpenCalls&email=klimek@ksi.mff.cuni.cz&action=getSuitableOpenCalls&cpv=30213000&expired=true`

This list of subscriptions can be viewed by using **`list`** action on the web service and processed using the **`process`** action. This call can be made regularly (e.g., by putting a script in **`cron`**), sending results, for example, daily to all registered emails.

There are four actions for the subscription web service. **`subscribe`** (Table 7) and **`unsubscribe`** (Table 8) actions have their parameters described in the respective tables. The **`list`** action lists all subscriptions. The **`clear`** action clears all subscriptions.

2.5 Deployment

To deploy the matchmaking web services on your own infrastructure follow these steps:

1. Deploy the project's WAR file on a Tomcat 7 server.[6]
2. Copy the **`cache`** and **`config`** folders to the root folder of your Tomcat (e.g. the **`tomcat`** folder which contains the **`webapps`** folder).
3. Set the file permissions for these two folders so that Tomcat can write into them.
4. In **`WEB-INF/web.xml`** set the **`sparql_public_query`** path to any SPARQL 1.1 compliant query endpoint.[7]
5. Load data into the instance of the RDF store exposed via the SPARQL query endpoint. The public contracts data must be modelled according to the Public Contracts Ontology.

[6] `http://tomcat.apache.org/tomcat-7.0-doc/`
[7] The prototype was tested with Apache Jena's Fuseki (`http://jena.apache.org/documentation/serving_data/`).

Field	Datatype	Description
beURI	URI	URI of the current business entity
name	string	legal name of the current business entity
place	string	place of the current business entity
score	float (percentage)	suitability of the current business entity to be a supplier for the given contract
contracts	integer	number of contracts, to which the current business entity supplied to
contractsSameCPV	integer	number of contracts, to which the current business entity supplied to, described by at least one of the CPV codes describing the input contract
volumeOfContracts	integer	total volume (price) of all contracts the current business entity supplied to
volumeOfContractsSameCPV	integer	total volume (price) of all contracts the current business entity supplied to that are described by at least one of the CPV codes used to describe the input contract
currency	string	currency of the price volume contracts
contractingAuthorities	integer	number of contracting authorities with which the current business entity made contracts
contractingAuthoritiesSameCPV	integer	number of contracting authorities, with which the current business entity made contracts that are described by at least one of the CPV codes used to describe the input contract

Table 4: Suitable suppliers web service output description

Parameter	Required	Default	Possible values	Description
cpv	yes		comma-separated list of 8-digit or 9-digit (with control digit) CPV codes	list of CPV codes that the calls for tenders should match
expired	no	false	boolean	When set to `true`, the results will also contain calls for tenders with deadline for submission of tenders in the past. This is meant to be used for experiments when the available data is not up to date and does not contain calls for tenders that are still open.
mainOnly	no	false	boolean	When set to `true`, only the calls for tenders having one of the CPV codes as their `pc:mainObject` will be returned. A call for tenders can have one `pc:mainObject` and multiple `pc:additionalObject` CPV codes assigned.
awarded	no	false	boolean	When set to `true`, the results will also contain calls for tenders that are already awarded to a supplier.
offset	no	0	integer	0-indexed offset of the first result to return
limit	no	10	integer or `all`	maximum number of results to return
sortField	no	publicationDate	`title`, `price`, `cpv`, `publicationDate`, `tenderDeadline`	Sort results (before applying offset and limit) according to the specified field.
sortOrder	no	desc	`asc` or `desc`	sort order: `asc` (ascending) or `desc` (descending)

Table 5: Suitable calls for tenders web service parameters

Field	Datatype	Description
contractURI	URI	URI of the current call for tenders
title	string	title of the current call for tenders
description	string	description of the current call for tenders
currency	string	currency of the current call for tenders
cpvs	list of strings	CPV codes describing the current call for tenders
publicationDate	date	date of publication of the call for tenders
tenderDeadline	date	deadline for submission of tenders to the current call

Table 6: Suitable calls for tenders web service output description

Parameter	Required	Possible values	Description
action	yes	actions of the Matchmaker service	identification of the web service for subscription
email	yes	email address	email to which the results should be sent
...	yes		parameters of the identified Matchmaker web service action (described above)

Table 7: Subscriptions web service subscribe parameters

Parameter	Required	Possible values	Description
action	yes	`unsubscribe`	specifies that the email address should be unsubscribed.
email	yes	email address	email to which the results should be sent

Table 8: Subscriptions web service unsubscribe parameters

3 References

[1] Klímek, Jakub; Knap, Tomáš; Mynarz, Jindřich; Nečaský, Martin; Svátek, Vojtěch. *LOD2 deliverable 9a.1.1: Framework for creating linked data in the domain of public sector contracts* [online]. Prague, 2012 [cit. 2013-07-11]. Available from WWW: http://static.lod2.eu/Deliverables/deliverable-9a.1.1.pdf

[2] Mynarz, Jindřich; Nečaský, Martin; Veselka, Jiří. *LOD2 deliverable 9a.1.2: Web application for filing public contracts* [online]. Prague, 2012 [cit. 2013-07-16]. Available from WWW: http://static.lod2.eu/Deliverables/D9a.1.2.pdf