LODDY

Contents

What this document is all about

This document is a companion for the DEMO distribution of the (still under development) LODDY RDF Publishing framework.

The project started in 2014 and, after a first release, its development has been slowed due to other projects with higher priority, still leaving final touches to be finalized. However, though with some limitations, LODDY can already be used for easily publishing RDF data through descriptive HTML pages. For this reason, we have packed up a demo, which is completely configured for working on a local machine, exposing data from the Agrovoc [1] SPARQL endpoint.

Though setup on Agrovoc, the DEMO can however be fully customized in order to point to a different SPARQL endpoint, to use different configurations to read and show data, and to adopt different web page templates.

Now, a few words about LODDY, and then more technical details about the demo and the framework.

What is Loddy?

Loddy is a lightweight system for supporting publication of Linked Open Data. The purpose is pretty much similar to that of Pubby[1]. So, what changes with respect to Pubby?

1) A more open system, mostly providing RDF publication facilities over a well known stack of web publication technologies (JSF[2], Facelets[3])
2) Clearer separation of concerns:

[1] http://wifo5-03.informatik.uni-mannheim.de/pubby/
[2] http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html
[3] https://facelets.java.net/nonav/docs/dev/docbook.html

a. system administrators may configure existing beans for data access, proper redirections etc..
b. RDF experts may configure another range of existing beans to obtain the desired representation of the data
c. Web Editors may focus on customizing pretty understandable templates in standard clean XHTML for the pages to be published
d. Optionally, developers may create new bean definitions (java classes extending the Loddy system classes, or implementing Loddy interfaces) implementing functionalities (or performing some processing of the data) which are not already available through the beans provided by the system

3) Additional possibilities, such as:
   a. More flexible page organization
   b. Different templates for different type of data (the categorization may be done upon very different filters, such as the form of the URI, or even by prefetching the data and checking the rdf:types of the resource).

Loddy does not need to be smart, it just aims at being easily configurable by people who know well the resources they are publishing. For instance, in order to show pages specifically rendered for different type of resources, in Agrovoc there is no need to check their rdf:type: if a resource is a skosxl:Label, then its localname has the form: "xl_<shortUUID>", so it suffices to apply a bean for page orchestration based on a regular expression. In other scenarios where the URI is not suggesting the type of a resource, this bean implementation may be replaced with others which perform the same task (page rerouting) but on the basis of different evidences.

Setup

In order to start using Loddy, users may start from its Agrovoc based demo, and then customize it to their SPARQL endpoint and specific publication needs.

Before using the Agrovoc DEMO, it is important to verify that the Agrovoc resource is up and responding. A quick check at the address: http://aims.fao.org/aos/agrovoc/c_1221 may confirm that the Agrovoc server is up and running.

The first installation step consists in deploying the web application file (Loddy.war) into an application server (es. Tomcat). A visit to the local page http://localhost:8080/Loddy/c_1221 should thus redirect (http 303) to http://localhost:8080/Loddy/c_1221.html, an HTML page describing the *calyx* resource.

If Loddy does not respond, check if the application server is listening on port 8080 (Tomcat default port). In negative case, change the localBaseURI property of the ConnectionBean in WEB-INF/faces-config.xml file (for further information, read the information about ConncectionBean in the following *Demo* section).

Loddy has been compiled with Java 7 and successfully tested on Tomcat 7.0.52 and 8.0.27.

Demo

The demo distributed with LODDY, offers a (very simple) web template for representing resources from Agrovoc, according to a given configuration of web templates (expressed in Facelets) and LODDY Java beans.

Agrovoc is a thesaurus modeled according to the SKOS and SKOS-XL vocabularies. skos:Concepts are thus linguistically represented through reified labels (skosxl:Label) which have a lexical representation expressed through the property skosxl:iteralForm.

Here is an example of an Agrovoc concept (and one of its lexicalizations) expressed in SKOS-XL:

agrovoc:c_1221	rdf:type	skos:Concept
agrovoc:c_1221	skosxl:prefLabel	agrovoc:xl_92349023
agrovoc:xl_92349023	rdf:type	skosxl:Label
agrovoc:xl_92349023	skosxl:literalForm	"Calyx"@en

In the demo, we allowed for the possibility to publish pages representing both concepts and labels, and have these pages organized according to different templates.

If the requested resource is a skos:Concept, the description is shown through the web page template provided in *concept.xhtml*: this is a two-tables template (Figure 1) representing the (multilingual) lexical information of a concept on the right, and all the other information on the left.

The left part of the page offers a two-column table containing information obtained through a SPARQL DESCRIBE of the requested resource. This table is filtered out (through a dedicated bean) of all the triples related to lexical information. The table on the right reassumes the lexical information. Here we can see the high customization capabilities that Loddy is offering: the right section (with a separate query) is fetching only the skosxl labels and shows them in a readable way, directly showing the literal form for each skosxl:Label (thus skipping the URI in the middle). At the same time, each literalform (column "label") is also a link, which links to the URI of the reified label.

Figure 1: skos:Concept description

The left table is obtained through a SPARQL DESCRIBE of the concept (using a GraphQueryBean), then a filter on the label is applied (GraphFilterBean) and finally the statements of the resulting graph are grouped (GraphGrouper).

The right table, shows information about skosxl labels of the requested resource. This is obtained with a TupleQueryBean which performs the following SELECT query that retrieves: type (pref, alt or hidden), URI, literal form and language tag of the required resource.

```
select ?type ?uri ?lexical ?language where {
        <#{connectionBean.resourceUri}> ?type ?uri.
        ?uri a <http://www.w3.org/2008/05/skos-xl#Label>.
        ?uri <http://www.w3.org/2008/05/skos-xl#literalForm> ?literalForm.
        bind (lang(?literalForm) as ?language)
        bind (str(?literalForm) as ?lexical)
}
```

The footer contains two links "As RDF/XML" and "As TURTLE" obtained by the DispatcherBean's method getLinkForFormat().

Figure 2: skosxl:Label description

If the requested resource is a skosxl:Label, the description is organized according to the label.xhtml template (Figure 2). This page contains a table obtained through a (non filtered, in this case) DESCRIBE query. As for the previous case, the result of the DESCRIBE is grouped using a GraphGrouper bean.

In order to customize LODDY for your needs, you may start from this demo and edit the file WEB-INF/faces-config.xml and the page templates concept.xhtml, label.xhtml and error.xhtml (N.B. do not edit content_negotiation.xhtml, redirect.xhtml and rdf.xhtml).

In the following pages, we provide some information about the existing beans. Please notice that this section is totally “work in progress”, as we are revising the class organization, and the terminology adopted.

Configuration Beans

ConnectionBean:
Manages the connection. By editing faces-config.xml, you can configure LODDY in order to connect to a sparql endpoint. This bean has three configurable managed-properties:

- sparqlEndpointURL: URL of the SPARQL endpoint;
- baseURI: baseURI of the dataset;
- localBaseURI: useful to rewrite the URL of the resources. During the testing process this should point to the installed Loddy instance, in production it should have the same value of the baseURI property.

```
<managed-bean>
        <managed-bean-name>connectionBean</managed-bean-name>
        <managed-bean-class>it.uniroma2.art.loddy.beans.ConnectionBean</managed-bean-class>
```

```
        <managed-bean-scope>application</managed-bean-scope>
        <managed-property>
                <property-name>baseURI</property-name>
                <property-class>java.lang.String</property-class>
                <value>http://aims.fao.org/aos/agrovoc/</value>
        </managed-property>
        <managed-property>
                <property-name>sparqlEndpointURL</property-name>
                <property-class>java.lang.String</property-class>
                <value>http://202.45.139.84:10035/catalogs/fao/repositories/agrovoc</value>
        </managed-property>
        <managed-property>
                <property-name>localBaseURI</property-name>
                <property-class>java.lang.String</property-class>
                <value>http://localhost:8080/Loddy/</value>
        </managed-property>
</managed-bean>
```

The ConnectionBean, through the getResourceUri() method, allows to get the URI of the resource currently being requested.

DispatcherBean:

This bean manages content negotiation and redirection to the description pages. It has one configurable managed-property:

- pageMapping: a map associating resource types to their respective pages describing the resources.

In the demo, skos:Concept are described through the concept.xhtml page, and skosxl:Labels are described through label.xhtml.
To define a mapping for a generic resource (i.,e. not already specified by the other mappings), you can use a map-entry with key "default".

```
<managed-bean>
        <managed-bean-name>dispatcherBean</managed-bean-name>
        <managed-bean-class>it.uniroma2.art.loddy.beans.DispatcherBean</managed-bean-class>
        <managed-bean-scope>application</managed-bean-scope>
        <managed-property>
                <property-name>pageMapping</property-name>
                <property-class>java.util.Map</property-class>
                <map-entries>
                        <key-class>java.lang.String</key-class>
                        <value-class>java.lang.String</value-class>
                        <map-entry>
                                <key>http://www.w3.org/2008/05/skos-xl#Label</key>
                                <value>label.xhtml</value>
                        </map-entry>
                        <map-entry>
                                <key>http://www.w3.org/2004/02/skos/core#Concept</key>
                                <value>concept.xhtml</value>
                        </map-entry>
                </map-entries>
        </managed-property>
        <managed-property>
                <property-name>connectionBean</property-name>
                <property-class>it.uniroma2.art.loddy.beans.ConnectionBean</property-class>
                <value>#{connectionBean}</value>
        </managed-property>
</managed-bean>
```

The DispatcherBean also provides the method getLinkForFormat(String format), which returns the address of the page that describes the current resource in different serialization format (available: rdf, ttl, n3, nt).

KnownNamespacesBean:

A bean that contains some well known mapping namespace-prefixes (currently: rdf, rdfs, owl, skos, skosxl and foaf). It has one managed-property:

- namespacePrefixMapping: allows to add a new entry in the namespace-prefix mapping

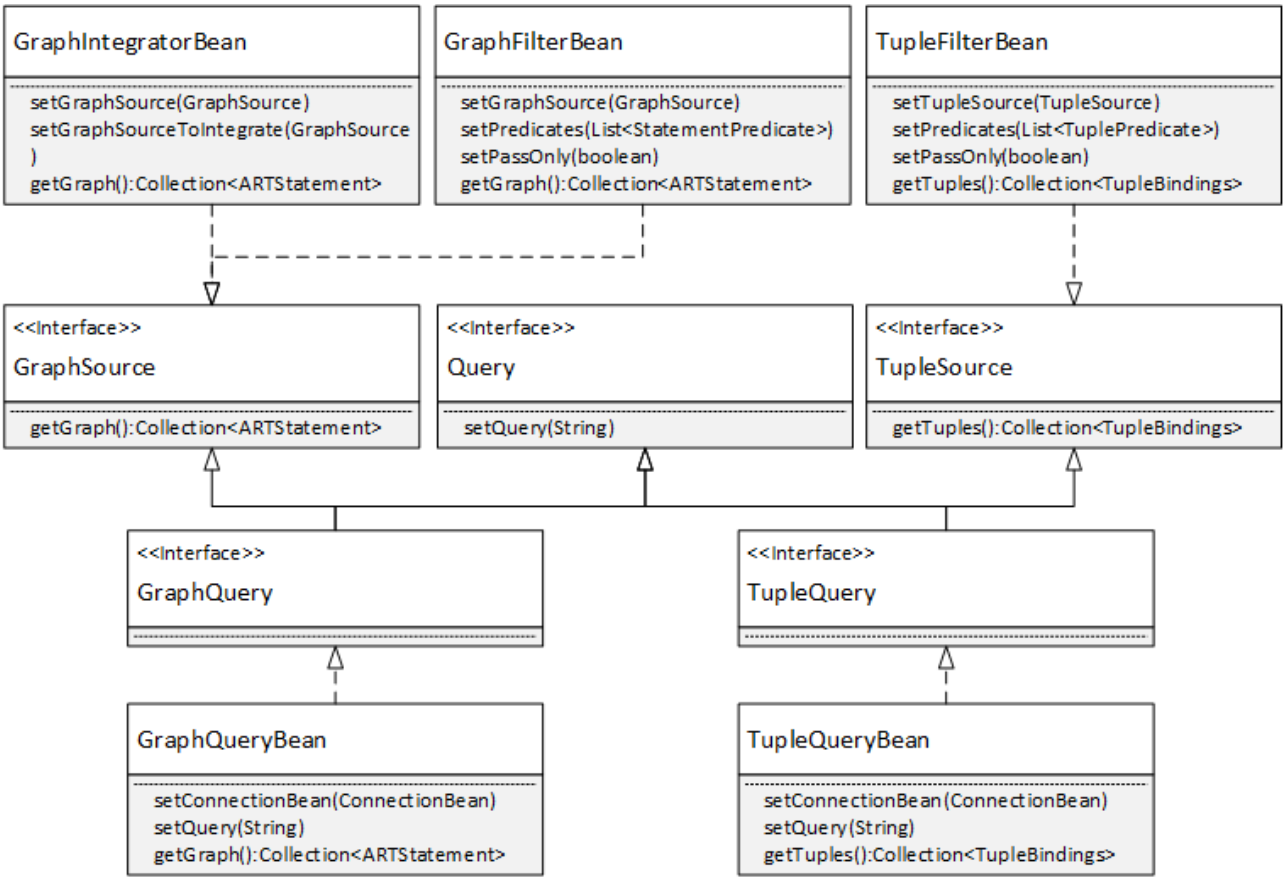
In the following example two entries are added: `http://rdfs.org/ns/void# - void` and `http://purl.org/dc/terms/ - dcterms`

```
<managed-bean>
	<managed-bean-name>knownNamespaces</managed-bean-name>
	<managed-bean-class>it.uniroma2.art.loddy.beans.KnownNamespacesBean</managed-bean-class>
	<managed-bean-scope>application</managed-bean-scope>
	<managed-property>
		<property-name>namespacePrefixMapping</property-name>
		<property-class>java.util.Map</property-class>
		<map-entries>
			<key-class>java.lang.String</key-class>
			<value-class>java.lang.String</value-class>
			<map-entry>
				<key>http://rdfs.org/ns/void#</key>
				<value>void</value>
			</map-entry>
			<map-entry>
				<key>http://purl.org/dc/terms/</key>
				<value>dcterms</value>
			</map-entry>
		</map-entries>
	</managed-property>
</managed-bean>
```

Content Beans

These are beans dealing with querying the connected dataset, filtering results and processing them in general.

Please, note that this schema is temporary, some methods will be factorized in common interfaces and maybe some interfaces will be deleted. It is just a temporary reference.



GraphQueryBean:

This bean performs graph queries (CONSTRUCT or DESCRIBE). It has two managed-properties that should be specified:

- connectionBean: to inject dependency on the ConnectionBean providing the sparql endpoint connection.
- query: specifies the query to perform. If not specified, the bean will perform a DESCRIBE on the requested resource.

```
<managed-bean>
	<managed-bean-name>graphQueryDescribe</managed-bean-name>
	<managed-bean-class>it.uniroma2.art.loddy.query.impl.GraphQueryBean</managed-bean-class>
	<managed-bean-scope>request</managed-bean-scope>
	<managed-property>
		<property-name>connectionBean</property-name>
		<property-class>it.uniroma2.art.loddy.beans.ConnectionBean</property-class>
		<value>#{connectionBean}</value>
	</managed-property>
</managed-bean>
```

TupleQueryBean:

A bean performing tuple queries (SELECT). It has two managed-properties:

- connectionBean: to inject a dependency on the ConnectionBean providing the sparql endpoint connection.
- query: specifies the query to perform.

In the following example, the query returns a collection of TupleBindings subject (?s) predicate (?p) object (?o), where the subject is the URI of the requested resource.

```
<managed-bean>
	<managed-bean-name>tupleQuery</managed-bean-name>
	<managed-bean-class>it.uniroma2.art.loddy.query.impl.TupleQueryBean</managed-bean-class>
	<managed-bean-scope>request</managed-bean-scope>
	<managed-property>
		<property-name>connectionBean</property-name>
		<property-class>it.uniroma2.art.loddy.beans.ConnectionBean</property-class>
		<value>#{connectionBean}</value>
	</managed-property>
	<managed-property>
		<property-name>query</property-name>
		<property-class>java.lang.String</property-class>
		<value>select ?s ?p ?o where {bind (&lt;#{connectionBean.resourceUri}&gt; as ?s) ?s
?p ?o}</value>
	</managed-property>
</managed-bean>
```

Note that the URI of the requested resource is retrieved from the ConnectionBean through the expression #{connectionBean.resourceUri}

GraphFilterBean:

Bean that filters a graph (statement collection). It has three managed-properties:

- predicates: collection of StatementPredicate. It represents the collection of predicate that should be verified by the filter.
- graphSource: graph to filter.
- passOnly: specifies if the filter should work in "pass only" mode (true) or "remove only" mode (false). In "pass only" mode, the resulting graph will contain all the statements that satisfy the StatementPredicate, otherwise, in "remove only" mode, it will contain all the statements of the original graph except those that satisfy the predicates.

```
<managed-bean>
	<managed-bean-name>graphFilterNoLabel</managed-bean-name>
	<managed-bean-class>it.uniroma2.art.loddy.processor.GraphFilterBean</managed-bean-class>
	<managed-bean-scope>request</managed-bean-scope>
	<managed-property>
		<property-name>predicates</property-name>
		<property-class>java.util.List</property-class>
		<list-entries>
			<value-class>it.uniroma2.art.loddy.predicate.StatementPredicate</value-class>
			<value>#{labelStatementPredicate}</value>
		</list-entries>
	</managed-property>
	<managed-property>
		<property-name>graphSource</property-name>
		<property-class>it.uniroma2.art.loddy.query.GraphSource</property-class>
		<value>#{graphQueryDescribe}</value>
	</managed-property>
	<managed-property>
		<property-name>passOnly</property-name>
		<property-class>boolean</property-class>
		<value>false</value>
	</managed-property>
</managed-bean>
```

ThreeBagsAdmittedValuesStatementPredicate:

A concrete implementation (currently the only one) of StatementPredicate. This bean implements a StatementPredicate to apply to a bean implementing the GraphSource interface. It has three managed-properties:

- subjectValues: list of admitted values of (rdf) subject.
- predicateValues: list of admitted values of (rdf) predicate.
- objectValues: list of admitted values of (rdf) object.

For each element (subject, predicate, object), the list of values admitted by it are evaluated in OR (i.e. the element is "satisfied" if at one of the values is matched). The predicate is satisfied if all of the three elements are satisfied.

If an element has no specified value, than that element is always satisfied (any value of the tested statement is accepted).

In the following example a StatementPredicate is configured so that it accepts triples concerning skos and skosxl labels. Note that this StatementPredicate was applied in "remove only" mode (passOnly=false) in the previous example about the GraphFilterBean.

```
<managed-bean>
	<managed-bean-name>labelStatementPredicate</managed-bean-name>
	<managed-bean-
class>it.uniroma2.art.loddy.predicate.impl.ThreeBagsAdmittedValuesStatementPredicate</managed-bean-
class>
	<managed-bean-scope>request</managed-bean-scope>
	<managed-property>
		<property-name>predicateValues</property-name>
		<property-class>java.util.List</property-class>
		<list-entries>
			<value-class>java.lang.String</value-class>
			<value>http://www.w3.org/2004/02/skos/core#prefLabel</value>
			<value>http://www.w3.org/2004/02/skos/core#altLabel</value>
			<value>http://www.w3.org/2004/02/skos/core#hiddenLabel</value>
			<value>http://www.w3.org/2008/05/skos-xl#prefLabel</value>
			<value>http://www.w3.org/2008/05/skos-xl#altLabel</value>
			<value>http://www.w3.org/2008/05/skos-xl#hiddenLabel</value>
		</list-entries>
	</managed-property>
</managed-bean>
```

Thus, in practice, all the triples having their predicate equal to any of the listed properties, are filtered out from the results.

TupleFilterBean (planned but not yet available):

A bean that filters a collection of tuple bindings. It has three properties:

- predicates: collection of TuplePredicate. It represents a collection of predicate that should be verified by the filter.
- tupleSource: tuple collection to filter.
- passOnly: specifies if the filter should work in "pass only" mode (true) or "remove only" mode (false). In "pass only" mode, the resulting tuple collection will contain all the tuples that satisfiy the TuplePredicate, otherwise, in "remove only" mode, it will contain the all the tuples of the original collection except those that satisfy the predicates.

SimpleTuplePredicate (planned but not yet available):

A bean that represents a predicate to be appled to a TupleSource. It has one property:

- filterMap: map that associates all admitted values to a binding.

GraphIntegratorBean:

Merges two graphs. It has two managed-property:

- graphSource: source graph.
- graphSourceToIntegrate: graph to merge with the source graph.

```
<managed-bean>
	<managed-bean-name>graphIntegratorBean</managed-bean-name>
	<managed-bean-class>it.uniroma2.art.loddy.processor.GraphIntegratorBean</managed-bean-class>
	<managed-bean-scope>request</managed-bean-scope>
	<managed-property>
		<property-name>graphSource</property-name>
		<property-class>it.uniroma2.art.loddy.query.GraphSource</property-class>
		<value>#{graphSourceName1}</value>
	</managed-property>
	<managed-property>
		<property-name>graphSourceToIntegrate</property-name>
		<property-class>it.uniroma2.art.loddy.query.GraphSource</property-class>
		<value>#{graphSourceName2}</value>
	</managed-property>
</managed-bean>
```

GraphGrouperBean:

A bean that groups statements in a graph by subject and then by predicate. Given a collection of statements it returns a collection of GroupedStatement. A GroupedStatement is a custom object that has two properties: "subject" (subject of statement) and "predObj", a map that contains predicate-objectList entries. GraphGrouperBean has one property:

- graphSource: source graph to group.

```
<managed-bean>
	<managed-bean-name>graphGrouper</managed-bean-name>
	<managed-bean-class>it.uniroma2.art.loddy.processor.GraphGrouperBean</managed-bean-class>
	<managed-bean-scope>request</managed-bean-scope>
	<managed-property>
		<property-name>graphSource</property-name>
		<property-class>it.uniroma2.art.loddy.query.GraphSource</property-class>
		<value>#{graphSourceName}</value>
	</managed-property>
</managed-bean>
```

RdfWriterBean:

A bean that serializes a graph according to different formats (rdf, n3, nt, ttl etc..). It has one property:

- graphQuery: graph to serialize.

```
<managed-bean>
	<managed-bean-name>rdfWriterBean</managed-bean-name>
	<managed-bean-class>it.uniroma2.art.loddy.beans.RdfWriterBean</managed-bean-class>
	<managed-bean-scope>request</managed-bean-scope>
	<managed-property>
		<property-name>graphQuery</property-name>
		<property-class>it.uniroma2.art.loddy.query.GraphQuery</property-class>
		<value>#{graphQueryDescribe}</value>
	</managed-property>
```

```
</managed-bean>
```

Beans not to change

FormatterBean:

This bean allows for URI rewriting. For instance, it rewrites the predicates of a statement in prefixed form (qnames). The sole existing implementation for qames has one property:

- knownNamespaces: reference to a KnownNamespacesBean from which namaespace-prefix mappings are obtained.

```
<managed-bean>
        <managed-bean-name>formatterBean</managed-bean-name>
        <managed-bean-class>it.uniroma2.art.loddy.beans.FormatterBean</managed-bean-class>
        <managed-bean-scope>application</managed-bean-scope>
        <managed-property>
                <property-name>knownNamespaces</property-name>
                <property-class>it.uniroma2.art.loddy.beans.KnownNamespacesBean</property-class>
                <value>#{knownNamespaces}</value>
        </managed-property>
        <managed-property>
                <property-name>connectionBean</property-name>
                <property-class>it.uniroma2.art.loddy.beans.ConnectionBean</property-class>
                <value>#{connectionBean}</value>
        </managed-property>
</managed-bean>
```

References

[1] C. Caracciolo, A. Stellato, A. Morshed, G. Johannsen, S. Rajbhandari, Y. Jaques e J. Keizer, «The AGROVOC Linked Dataset,» *Semantic Web Journal,* vol. 4, n. 3, p. 341–348, 2013.