5
The LDC Mediator

This chapter presents the architecture of the LDC Mediator. Section 5.1 argues for the advantages of using the REST Principles for consuming linked data cubes. Section 5.2 presents the components of the LDC Mediator. Section 5.3 describes the implementation of the mediator, including the platform, language, and tools used to build the mediator. This section also presents each package and its main java classes. Finally, Section 5.4 illustrates, through one of these Brazilian datasets collected, the process of consuming data cube metadata, as well as the process of consuming data cube observations.

5.1
Combining REST and Linked Data

The REST Principles are in a sense related to the Linked Data Principles in terms of the concept of resource:

- Resources have a unique identifier (URI) that must be dereferenceable through HTTP;
- Resources are interlinked, and, by following those links, new resources can be discovered.

However, the Linked Data Principles are only concerned with the publication and retrieval of data. The access to large collections of RDF data is usually done through a SPARQL endpoint. RESTful Web services in turn have no endpoints. They only provide a collection of resource URIs and operations to access and change the state of the resources. REST requires a uniform interface, a set of methods with known semantics that access and change the state of the resources. These methods are external to the resources, which can have multiple representations, and are invoked by HTTL messages sent to the server.

Therefore, in this context it is intuitive to combine Linked Data with REST Principles to allow data cube access and manipulation (Stadtmüller and Harth 2012). The RDF model is also flexible enough to support multiple representations, which capture the resource's current state (for instance XML or JSON).

5.2
The LDC Mediator components

Figure 21 shows the architecture of the mediator, which mediates access to the underlying statistical relational databases and exposes data cubes to the applications through a RESTful API.

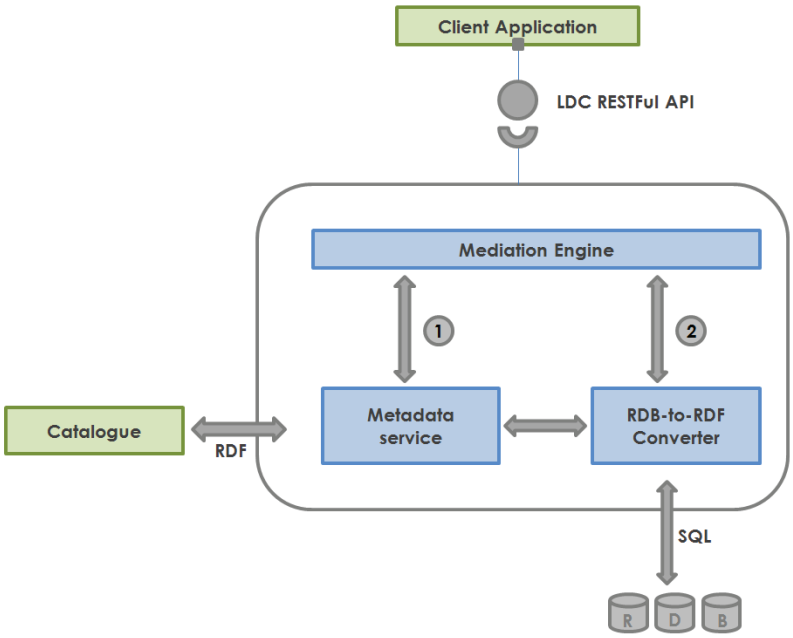

Figure 21: LDC Mediator components

The LDC Mediator has the following components: a LDC RESTful API, a Mediation Engine, a Metadata service and a RDB-to-RDF Converter. The *LDC RESTful API* is the component of the mediator that exposes the data cubes through a Web service implemented using the REST Principles (Fielding and Taylor 2002). The *Mediator Engine* is the component that receives a request from a Client Application and chooses how to process the request, depending on its type. The *Metadata Service* is the module responsible for retrieving data cube metadata requested by the Mediator Engine. These metadata include the cube dimensions,

the measure and also the connection information used to access the underlying relational database. The *RDB-to-RDF Converter* is the component used to generate the RDF triples that represent the data cube observations from the relational databases.

5.3 Implementation of the LDC Mediator

The LDC Mediator was developed using the following platforms, languages and tools:

- Windows 7 Starter SP1 32 bits
- Java Plataform, Enterprise Edition 7 SDK[27] (version 1.7.0_25)
- NetBeans IDE 7.2.1[28]
- My Sql Server 5.5[29]
- Toad for MySql 6.3 Freeware [30]
- Virtuoso open source edition [31]
- DB2Triples – RDB2RDF Antidot implementation[32]
- Jersey RESTful Web Services in Java[33]
- Astah Professional 6.6.4/41775[34]

In addition to the documentation of each of these frameworks, some documents were essential in the implementation of the LDC Mediator:

- Stack Overflow, a question and answer Web site for programmers[35]
- GUJ forum for Java programmers[36]
- The guide to install and configure Virtuoso on Windows[37]
- Examples of using Jena to manipulate RDF graphs[38]
- "My first mapping from RDB to RDF with R2RML", Ivan Herman[39]

[27] http://www.oracle.com/technetwork/java/javaee/javaee7sdk-install-1957708.html
[28] https://netbeans.org/
[29] http://dev.mysql.com/downloads/mysql/
[30] http://www.quest.com/toad-for-mysql/
[31] http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSDownload
[32] https://github.com/antidot/db2triples
[33] https://jersey.java.net/
[34] http://astah.net/editions/community
[35] http://stackoverflow.com/
[36] http://www.guj.com.br/forums/list.java
[37] http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSUsageWindows
[38] http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtJenaProvider

The LDC Mediator architecture is divided into three packages, according to its components (see Figure 21): the Mediator Engine package, the Metadata Service package and the RDB-to-RDF package. These packages are detailed in the following sections (the test package, the exception handling, and other auxiliary classes were omitted).

5.3.1. The Mediator Engine package

The Mediator Engine package contains classes that directly deal with client requests, as well as classes that represent the REST resources. The main class of this package is the `MediatorService.java` class which has attributes and methods used to interact with the other packages. It also has two classes that represent a data cube resource (`DataCubeResource.java`) and a list of data cubes resources (`DataCubesListResource.java`), as indicated in Figure 22.

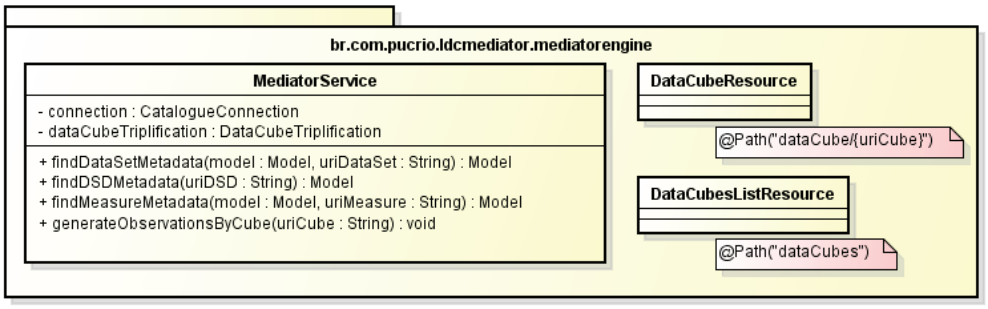

Figure 22: Mediator Engine package

As previously mentioned in Section 2.6.2, data cube metadata is divided into three parts: the dataset definition, the structure definition and the measure(s) definition. In order to simplify cube metadata consumption, the `MediatorService.java` class has three independent methods: the first method (`findDataSetMetadata`) gets the dataset definition part of the cube definition; the second method (`findDSDMetadata`) gets the data structure definition (DSD) and the third method (`findMeasureMetadata`) gets the metadata of the measure(s).

In order to represent the REST resources, this package uses the Java API for RESTful Services, called JAX-RS[40]. The reference implementation for this

[39] http://ivan-herman.name/2010/11/02/my-first-mapping-from-rdb-to-rdf-using-r2rml/

[40] http://jax-rs-spec.java.net

specification is an open source framework known as Jersey[41], which basically contains a REST server and a REST client. On the server side, Jersey uses a servlet which scans predefined classes to identify RESTful resources. Table 4 presents the most important JAX-RS annotations and their descriptions.

Table 4: The JAX-RS most important annotations

JAX-RS annotation	Description
@PATH(path)	Sets the path to base URL + /path.
@POST	Indicates that the following method will answer to a HTTP POST request
@GET	Indicates that the following method will answer to a HTTP GET request
@Produces(MediaType.TEXT_PLAIN [, more-types])	Defines which MIME type is delivered by a method annotated with @GET. In the example text ("text/plain") is produced. Other examples would be "application/xml" or "application/json".
@Consumes(type [, more-types])	Defines which MIME type is consumed by this method.
@PathParam	Used to inject values from the URL into a method parameter. This way you inject for example the ID of a resource into the method to get the correct object.

The `DataCubeResource.java` and the `DataCubeListResource.java` classes represent the predefined data classes (resources) that the Jersey scans automatically. Figure 23 shows the annotations used to implement the first resource when a client application requests a data cube metadata, for instance `HTTP GET http://host:port/LDC_Mediator/dataCube/dataset-residents`. The `MediaType.APPLICATION_OCTET_STREAM` type (line 39) is a generic mime type for non-text data and is part of the HTTP content negotiation (see Section 2.2) from the server to the clients.

```
30  /**
31   * REST Web Service
32   *
33   * @author Livia Ruback
34   */
35  @Path("dataCube/{uriCube}")
36  public class DataCubeResource {
37
38      @GET
39      @Produces(MediaType.APPLICATION_OCTET_STREAM)
40      public Response getDataCube(@PathParam("uriCube") String uriCube) {...}
```

Figure 23: The Jersey implementation of the Data Cube resource

[41] http://jersey.java.net/

5.3.2. The Metadata Service package

The Metadata Service package, shown in Figure 24, includes the class that connects directly to the Catalogue stored in the Virtuoso server (`CatalogueConnection.java`). This package also contains the auxiliary class (`SparqlQueries.java`) that builds the SPARQL queries needed to access the data cube metadata stored in the Catalogue. This class is also composed of some static attributes (omitted from the figure), which represent all vocabulary namespaces used to build the queries.

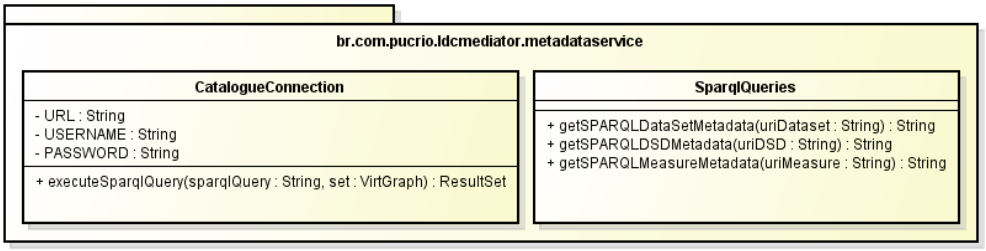

Figure 24: Metadata Service package

The main method of `CatalogueConnection.java` is `executeSparqlQuery(String sparqlQuery, VirtGraph set)`, presented in Figure 25. It uses classes from the Jena framework (`Query.java`, `QueryFactory.java` and `ResultSet.java`) and from the Virtuoso Jena Provider framework (`VirtGraph.java`, `VirtuosoQueryExecution.java`, `VirtuosoQueryExecutionFactory.java`). It receives a generic SPARQL query as parameter and a graph of triples, it returns a list of results and it is always used when a SPARQL query is executed against the Catalogue.

```
32          public ResultSet executeSparqlQuery(String sparqlQuery, VirtGraph set){
33                  try{
34                          Query sparql = QueryFactory.create(sparqlQuery);
35
36                          VirtuosoQueryExecution vqe = VirtuosoQueryExecutionFactory.create(sparql, set);
37
38                          ResultSet results = (ResultSet) vqe.execSelect();
39
40                          return results;
41                  }
42                  catch(Exception ex) {
43                          System.out.println("Error on method executeSparqlQuery: " + ex.getMessage());
44                          return null;
45                  }
46          }
```

Figure 25: The generic method to execute a SPARQL

5.3.3. The RDB-to-RDF package

The RDB-to-RDF package includes the classes used to convert the data cubes stored in the relational databases to RDF triples using R2RML mapping files. The two main classes of the package are: the `DataCubeTriplification.java` and the `DB2TriplesParameters.java`, shown in Figure 26.

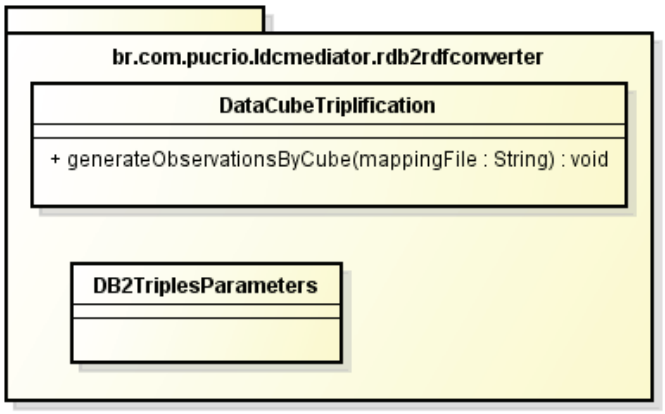

Figure 26: RDB-to-RDF Converter package

In order to read and interpret R2RML mapping files, the `DataCubeTriplification.java` employs the db2triples library[42], a software tool for extracting data from relational databases and loading data into RDF. It implements R2RML and Direct Mapping standards defined by the RDB2RDF working group[43]. This library accepts some parameters to configure the type of RDB-to-RDF conversion, listed in Table 5.

Table 5: DB2Triples parameters

Parameter	Description
-b	Database name
-d	Driver to use (default:com.mysql.jdbc.Driver)
-f	Force loading of existing repository (without remove data)
-i	Base URI (default:http://foo.example/DB/)
-l	Database URL (default :jdbc:mysql://localhost/)
-m	Mandatory conversion mode, 'r2rml' for R2RML and 'dm' for Direct Mapping

[42] http://www.w3.org/2001/sw/wiki/Db2triples
[43] http://www.w3.org/2001/sw/rdb2rdf/

-n	Native store output directory
-o	Output RDF filename (default : output)
-p	Database password
-q	Transformed graph output file (optional if sparql option is not specified, default : sparql_output otherwise)
-r	R2RML config file used to convert relational database into RDF terms.
-s	Sparql transform request file (optional)
-t	RDF syntax output format ('RDFXML', 'N3', 'NTRIPLES' or 'TURTLE')
-u	Database user name
-v	Version of norm to use (1 = Working Draft 20 September 2011 (default), 2 = Working Draft 23 March 2011)

These DB2Triples parameters are represented in the `DB2TriplesParameters.java` class by the static attributes that are in fact used to triplify the data cubes: `-m` (conversion mode, "r2rml"), `-u` (user), `-p` (password), `-b` (database name), `-t` (RDF syntax output format) and `-r` (the path of the R2RML mapping file). The conversion mode parameter is used to inform the RDB-to-RDF conversion approach: the Direct Mapping (see section 2.7.1) or the R2RML Mapping (see section 2.7.2).

5.4 Consuming data cubes with the LDC Mediator

This section provides an example that illustrates how the LDC Mediator works.

5.4.1. Datasets collected

A small data set from SIDRA (Sistema IBGE de Recuperação automática)[44] was collected to test the LDC Mediator. SIDRA is a Brazilian Government Repository which stores demographic census data. Initially, three datasets were extracted:

[44] http://www.sidra.ibge.gov.br/

- *[1] Foreigners living in Brasil*[45], which is broken down by condition (naturalized or not), gender, living situation (urbana or rural) and country of birth - containing 828 observations;
- *[2] Residents of Brazil*[46], which is broken down by race, sex, living situation, age group and age(year) - containing 872 observations;
- *[3] Residents of Brazil by Religion*[47], broken down by religion, year and region of Brazil - encompassing 864 observations.

These three datasets were extracted in CSV format and transformed into three different relational star schemas, each one containing one fact table related to their dimensions respectively. Figure 27 shows the star schema related to second dataset: residents of Brazil. Section 5.4.3 presents the data cube description related to this star schema.

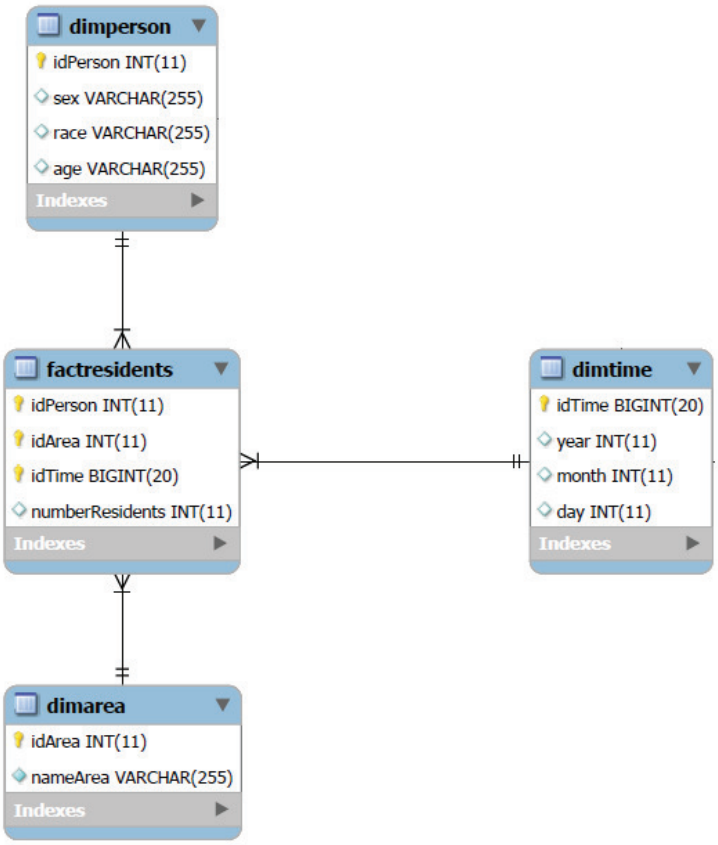

Figure 27: A residents star schema

[45] http://www.sidra.ibge.gov.br/bda/tabela/listabl.asp?z=cd&o=2&i=P&c=2093
[46] http://www.sidra.ibge.gov.br/bda/tabela/listabl.asp?z=cd&o=2&i=P&c=2093
[47] http://www.sidra.ibge.gov.br/bda/tabela/listabl.asp?z=cd&o=2&i=P&c=137

5.4.2. The RESTful Web service definition

The LDC Mediator provides access to the data cubes through a RESTful Web service, following the REST Principles. The RESTful HTTP methods implemented by the component are shown in Table 6. The first HTTP method offers a list of all cubes stored in the Catalogue. The second HTTP method provides the data cube metadata. Finally, the third HTTP method retrieves all the individual observations related to the cube.

Table 6: LDC RESTful API methods

Concept	HTTP Method	URI Template
Data Cubes List	GET	`http://host:port/LDC_Mediator/dataCubes`
Data Cube Metadata By Id	GET	`http://host:port/LDC_Mediator/dataCube/{uriDataCube}`
Observations by cube	GET	`http://host:port/LDC_Mediator/observations/{uriDataCube}`

By requesting the first HTTP method (`GET http://host:port/LDC_Mediator/dataCubes`) the client receives a list containing all cubes available and their descriptions, depicted in Figure 28. The list also contains links to the other HTTP methods in order to request cube metadata and cube observations. The second and the third methods will be exemplified in sections 5.4.3 and 5.4.4 respectively.

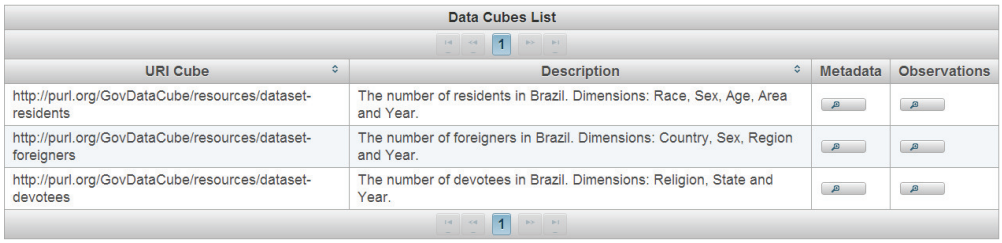

Figure 28: List of cubes

5.4.3. Requesting a cube metadata

The second HTTP method requests the cube metadata and is shown in Flow 1 of Figure 21 through the interaction between the components *Mediator Engine*

and *Metadata Service*. Following the REST Principles, the HTTP request `GET http://host:port/LDC_Mediator/dataCube/dataset-residents` returns the state of the resource "data cube residents", which in this case is the metadata of the cube, presented in Figure 29.

```
1  @prefix rdf:            <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
2  @prefix rdfs:           <http://www.w3.org/2000/01/rdf-schema#>.
3  @prefix xsd:            <http://www.w3.org/2001/XMLSchema#>.
4  @prefix qb:             <http://purl.org/linked-data/cube#>.
5  @prefix ex-property:    <http://purl.org/GovDataCube/properties/> .
6  @prefix ex-resource:    <http://purl.org/GovDataCube/resources/> .
7  @prefix sdmx-attribute: <http://purl.org/linked-data/sdmx/2009/attribute#>.
8  @prefix sdmx-measure:   <http://purl.org/linked-data/sdmx/2009/measure#>.
9
10
11 #================================================================================================
12 # Cube Residents
13 #================================================================================================
14
15 ex-resource:dataset-residents a qb:DataSet;
16     rdfs:label "Number of residents";
17     rdfs:comment "The number of residents in Brazil. Dimensions: Race, Sex, Age, Area and Year.";
18     qb:structure ex-resource:dsd-residents.
19
20 ex-resource:dsd-residents a qb:DataStructureDefinition;
21     # The dimensions
22     qb:component [qb:dimension ex-resource:dimRace];
23     qb:component [qb:dimension ex-resource:dimSex];
24     qb:component [qb:dimension ex-resource:dimAge];
25     qb:component [qb:dimension ex-resource:dimYear];
26     qb:component [qb:dimension ex-resource:dimArea];
27
28     # The measure
29     qb:component [qb:measure ex-property:numberResidents];
30     # The attributes
31     qb:component [qb:attribute sdmx-attribute:unitMeasure; qb:componentAttachment qb:DataSet;].
32
33 ex-property:numberResidents a rdf:Property, qb:MeasureProperty;
34     rdfs:label "number of residents";
35     rdfs:subPropertyOf sdmx-measure:obsValue;
36     rdfs:range xsd:integer .
```

Figure 29: Return of the cube metadata request

In order to simplify metadata consumption, the *Metadata Service* divides the metadata into three parts (the three figure frames) and groups the results to return the complete cube metadata (see Section 2.6.2). The first part (lines 15-18) creates a resource of type `qb:DataSet` (line 15) and represents the entire data set. The second part (lines 20-31) defines the structure of one or more datasets (a `qb:DataStructureDefinition` resource type). It also defines the dimensions, attributes, and measures of the structure. Figure 6 reveals that the components of a data structure definition can be of the following types: `qb:DimensionProperty`, `qb:AttributeProperty` or `qb:MeasureProperty`. The third part (lines 33-36) defines the measure corresponding to the phenomenon being observed that will give the value of each observation.

There are two approaches to finding the predicates and objects related to a specific resource. To exemplify each approach, the first part of the cube metadata presented in Figure 29 is considered. The first approach is a SPARQL query,

which explicitly informs a group pattern that must match the graph stored in Virtuoso triplestore[48], shown in Figure 30.

```
PREFIX qb:              <http://purl.org/linked-data/cube#>
PREFIX rdfs:            <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf:             <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ex-resource:     <http://purl.org/GovDataCube/resources/>

select ?comment ?label ?dsd
where {
  ex-resource:dataset-residents rdf:type qb:DataSet.
  ex-resource:dataset-residents rdfs:label ?label.
  ex-resource:dataset-residents rdfs:comment ?comment.
  ex-resource:dataset-residents qb:structure ?dsd.
}
```

Figure 30: SPARQL Query to return a metadata cube subset

Alternatively, instead of a static SPARQL query, one can dynamically search starting from a certain resource (identified by its URI), as presented in Figure 31. The piece of code shown in Figure 31 uses the Jena Framework[49], an API for reading, processing and writing RDF data, among other purposes.

```
63          public Model findDataSetMetadataPart1(Model model, String uriDataSet){
64              Model modelOutput = ModelFactory.createDefaultModel();
65
66              Resource resource = model.getResource(uriDataSet);
67              StmtIterator iter = resource.listProperties();
68              while (iter.hasNext()){
69                  Statement nextStatement = iter.nextStatement();
70                  // triple's property
71                  Property property = nextStatement.getPredicate();
72
73                  // triple's object
74                  RDFNode object = nextStatement.getObject();
75
76                  // adding triples to model
77                  modelOutput.add(resource, property, object);
78              }
79              return modelOutput;
80          }
```

Figure 31: Searching dynamically triples related to a resource

In Jena[50], the class used to represent a single triple is `Statement`. The methods for extracting the other components of a statement starting from the subject (represented by a `Resource` class element) are: `getPredicate()` that returns a `Property` and `getObject()` that returns a `RDFNode`.

[48] http://virtuoso.openlinksw.com/
[49] http://jena.apache.org/
[50] http://jena.apache.org/documentation/rdf/

In most cases, the Metadata Service implements the dynamic approach to find other resources related to a resource to take advantage of the flexibility of the RDF model.

5.4.4. Requesting cube observations

The third HTTP method

`GET http://host:port/LDC_Mediator/observations/{idCube}`

retrieves all the individual observations related to the cube. It has the unique identifier `{idCube}` and is shown in Flow 2 of Figure 24 through the interaction between the *Mediator Engine* and *RDB-to-RDF Converter* components.

Using the same residents cube example, the *Mediator Engine* receives the

`GET http://host:port/LDC_Mediator/observations/dataset-residents`

request and finds the R2RML mapping matching with this cube, which is depicted in Figure 32.

The URI of the triple map is `ex-resource:TriplesMapFactResidents` (line 12). Lines 14 to 24 create a view that maps each result row to a number of RDF triples, which are defined by predicate object maps. These, in turn, consist of predicate maps and object maps (or referencing object maps). Figure 32 presents only two dimensions (the others were omitted).

```
@prefix ex-property:    <http://purl.org/GovDataCube/properties/>.
@prefix ex-resource:    <http://purl.org/GovDataCube/resources/>.
@prefix ex-class:       <http://purl.org/GovDataCube/classes/>.
@prefix sdmx-attribute: <http://purl.org/linked-data/sdmx/2009/attribute#>.
@prefix qb:             <http://purl.org/linked-data/cube#>.
@prefix rr:             <http://www.w3.org/ns/r2rml#>.
@prefix rdf:            <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

# R2RML mapping to generate the residents cube observations from the fact table
ex-resource:TriplesMapFactResidents a rr:TriplesMap;
    d2rq:dataStorage ex-resource:databaseResidents;
    rr:logicalTable [ rr:sqlQuery """
                            SELECT DISTINCT
                              idPerson,
                              idArea,
                              idTime,
                              numberResidents
                            FROM
                              factResidents  """; ];
     rr:subjectMap [
        rr:template "http://purl.org/GovDataCube/resources/Observations/
                                {idPerson}_{idArea}_{idTime}_{numberResidents}";
        rr:class qb:Observation;
    ];
    rr:predicateObjectMap [
        rr:predicate qb:dataSet;
        rr:objectMap [rr:constant ex-resource:dataset-residents];
    ];
    rr:predicateObjectMap [
        rr:predicate ex-resource:dimSex;
        rr:objectMap [
            rr:parentTriplesMap ex-resource:TriplesMapSex;
            rr:joinCondition [
                rr:child "idPerson";
                rr:parent "idPerson";
            ];
        ];
    ];
    rr:predicateObjectMap [
        rr:predicate ex-resource:dimAge;
        rr:objectMap [
            rr:parentTriplesMap ex-resource:TriplesMapAge;
            rr:joinCondition [
                rr:child "idPerson";
                rr:parent "idPerson";
            ];
        ];
    ]; {other dimensions ommited}
    rr:predicateObjectMap [
        rr:predicate ex-property:numberResidents;
        rr:objectMap [rr:column "numberResidents"];
    ];
    rr:predicateObjectMap [
        rr:predicate sdmx-attribute:unitMeasure;
        rr:objectMap [rr:constant <http://dbpedia.org/ontology/Person>];
    ].
```

Figure 32: R2RML mapping file to the residents cube

Figure 33 shows an observation (1 out of 6480 observations) generated from the R2RML file described in Figure 32. Nine triples were generated for each observation. The first triple indicates that the resource is of the `qb:Observation` type (line 11). The second triple links the observation with its corresponding dataset (line 12). The third triple informs the class that qualifies the interpretation

of the observed value, `http://dbpedia.org/ontology/Person` (line 13). This class belongs to an external dataset (DBpedia) and enriches the data cube representation. The fourth triple presents the measure of the observation, `ex-property:numberResidents` (see the measure definition in Figure 29) and its value (line 14). Finally, the other triples link this observation to the dimension values (lines 21 to 34).

```
1  @prefix xsd:            <http://www.w3.org/2001/XMLSchema#>.
2  @prefix qb:             <http://purl.org/linked-data/cube#>.
3  @prefix ex-class:       <http://purl.org/GovDataCube/classes/>.
4  @prefix ex-property:    <http://purl.org/GovDataCube/properties/> .
5  @prefix ex-resource:    <http://purl.org/GovDataCube/resources/> .
6  @prefix ex-observation: <http://purl.org/GovDataCube/Observations/> .
7  @prefix sdmx-dimension: <http://purl.org/linked-data/sdmx/2009/dimension#>.
8  @prefix sdmx-attribute: <http://purl.org/linked-data/sdmx/2009/attribute#>.
9  @prefix dbpedia-ont:    <http://dbpedia.org/ontology/>.
10
11 ex-observation:201_1_2_6931 a qb:Observation ;
12     qb:dataSet ex-resource:dataset-residents ;
13     sdmx-attribute:unitMeasure dbpedia-ont:Person ;
14     ex-property:numberResidents "6931"^^xsd:integer ;
15     ex-resource:dimArea <http://purl.org/GovDataCube/resources/Area/03e5d717932176817f3cf0a794094c34> ;
16     ex-resource:dimYear <http://purl.org/GovDataCube/resources/Year/08f90c1a417155361a5c4b8d297e0d78> ;
17     ex-resource:dimAge <http://purl.org/GovDataCube/resources/Age/8d02c4cafd598c790c2555cdac8088c3> ;
18     ex-resource:dimRace <http://purl.org/GovDataCube/resources/Race/8ec4466b9f817546ff3cafa8341cf356> ;
19     ex-resource:dimSex <http://purl.org/GovDataCube/resources/Sex/97e8578f2aab557cf86014abb07db01a> .
20
21 <http://purl.org/GovDataCube/resources/Area/03e5d717932176817f3cf0a794094c34> a ex-class:Area ;
22     <http://www.w3.org/2000/01/rdf-schema#label> "Urbana"@pt .
23
24 <http://purl.org/GovDataCube/resources/Year/08f90c1a417155361a5c4b8d297e0d78> a ex-class:Year ;
25     <http://www.w3.org/2000/01/rdf-schema#label> "2000"^^<http://www.w3.org/2001/XMLSchema#string> .
26
27 <http://purl.org/GovDataCube/resources/Age/8d02c4cafd598c790c2555cdac8088c3> a ex-class:Age ;
28     <http://www.w3.org/2000/01/rdf-schema#label> "70 a 79 anos"@pt .
29
30 <http://purl.org/GovDataCube/resources/Race/8ec4466b9f817546ff3cafa8341cf356> a ex-class:Race ;
31     <http://www.w3.org/2000/01/rdf-schema#label> "Indígena"@pt .
32
33 <http://purl.org/GovDataCube/resources/Sex/97e8578f2aab557cf86014abb07db01a> a ex-class:Sex ;
34     <http://www.w3.org/2000/01/rdf-schema#label> "Mulheres"@pt .
```

Figure 33: Example of an observation

5.5 Summary

This chapter presented the architecture of the LDC Mediator. The components of the mediator were presented: the Mediator Engine, which provides a RESTful interface to access the data cubes; the Metadata Service, which connects with the Catalogue and manages the building and execution of the SPARQL queries; and the RDB-to-RDF Converter, which converts the star schemas stored in relational databases into RDF triples using R2RML mapping files. This chapter also discussed the implementation aspects, including the technologies, IDE's and frameworks used to build the LDC Mediator; as well as the packages created for each of these three components and their main classes.

Finally, an example was presented to illustrate data cube consumption, including cube metadata request and a cube observations request.