Combining and Uniting Business Intelligence with Semantic Technologies
Acronym: CUBIST Project No: 257403
Small or Medium-scale Focused Research Project FP7-ICT-2009-5 Duration: 2010/10/01-2013/09/30

Semantic ETL from structured data sources v.2

Abstract: This deliverable will provide approaches for automated or semi-automated mapping of structured enterprise data sources (databases) to the RDF conceptual model, so that they can be integrated with the RDF triple store.

Type	Report
Document ID:	CUBIST D2.2.2.
Work package:	WP2
Leading partner:	SAP
Author(s):	Katja Pfeifer (SAP)
Dissemination level:	PU
Status:	final
Date:	26 October 2012
Version:	1.0

Versioning and contribution history

Version	1. Description	Contributors
0.1	Draft (Introduction, Semantic ETL, Semantic ETL from structures data sources in CUBIST)	Katja Pfeifer (SAP)
0.2	RDB2RDF + tools update	Alex Simov (ONTO)
0.3	Moved general Semantic ETL into Introduction, Extended information on Semantic ETL in CUBIST	Katja Pfeifer (SAP)
0.4	Conclusion	Katja Pfeifer (SAP)
0.5	Overall reading + minor corrections	Alex Simov (ONTO)
1.0	Full Review	Axel Schröder (SAP) Cassio Melo (CRSA)

Table of contents

1 Introduction

This deliverable gives updates on the Semantic Extract-Transfer-Load (ETL) process from structured sources with respect to deliverable D2.2.1. It focuses on updates of techniques and implementation for the Semantic ETL process. Furthermore, it motivates the tools used for the Semantic ETL process for the three CUBIST use cases based on the use case specific data sets (see use case specific data interface design documents D7.2.1, D8.2.1 and D9.2.1) and the specific requirements. Additionally, the deliverable will give a short summary of the concrete Semantic ETL processes for the three CUBIST use cases. Practical details of how data is extracted from sources and loaded into the integrated triple store have been provided in D2.3.1 for M18 of CUBIST, and will be provided in D2.3.2 for M30 of CUBIST.

The term Semantic ETL refers to the ETL term used in "classical" BI. It recaps the process of federating data from various sources into a data warehouse by extracting (i.e. pulling the data from the sources), transforming (i.e. cleansing and enriching the data –e.g. with time stamps- and transforming it in order to suit the overall schema of the data warehouse) and loading the data (i.e. finally storing the transformed data in the data warehouse).

The Semantic ETL process for structured data sources differs from traditional ETL in the following points:

- The data is not transformed and mapped to database tables, but to a graph structure.
- The federated data is not stored in a Data Warehouse, but in a Triple Store instead.

The general process of Semantic ETL is illustrated in Figure 1. The extraction and transformation step differs depending on the data sources and the target structure. We will present this two steps separated for all three use cases. Outcome of this two steps are triple files of different formats (RDF/XML, N3, N-Triple, etc.) that have to be loaded into the triple store afterwards.

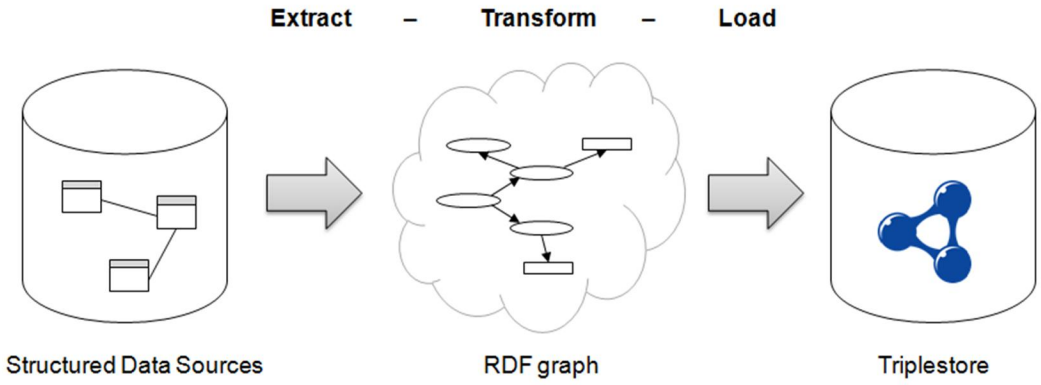

Figure 1: Semantic ETL Process

In Deliverable D2.2.1 we analysed techniques and tools for the Semantic ETL process – especially for the extraction and transformation process. In this deliverable, we will give an update on such tools and discuss the application for the Semantic ETL process for the three CUBIST use cases.

This deliverable is structured as follows: In Section 2 we review the updates on Relational Databases to RDF (RDB2RDF) techniques especially in context with W3C recommendations. Section 3 provides updates on specific tools/ implementations for Semantic ETL from structured data sources with

respect to D2.2.1. The application and the proper selection of the right tools for each CUBIST use case is discussed in Section 4. Finally, Section 0 concludes the document.

2 W3C RDB2RDF Update

By the time of preparing this document, two W3C RDB2RDF[1] documents became official recommendations. This means that the specifications have gone through extensive community review and revision and that *R2RML*[2] and *Direct Mapping*[3] are now considered stable enough for wide-spread distribution. In the following two sections we give an overview and examples of those two standards.

2.1 Direct Mapping

The direct mapping defines an RDF Graph representation of the data in a relational database. The direct mapping takes as input a relational database (data and schema), and generates an RDF graph that is called **direct graph**.

Foreign keys in relational databases establish a reference from any row in a table to exactly one row in a (potentially different) table. The direct graph conveys these references, as well as each value in the row. Note that the direct graph structure is predefined by the database schema and cannot be adjusted for custom RDF Graph structures. The latter is supported by the R2RML mapping language described in the next section.

2.1.1 Example

A simple database of two tables will demonstrate how direct (RDF) graph is generated. The two tables have private key column **ID** and one of them (*People*) refers the other one (*Addresses*) via foreign key (*addr*)

People

PK		→ *Address(ID)*
ID	**fname**	**addr**
7	Bob	18
8	Sue	**NULL**

Addresses

PK		
ID	**city**	**state**
18	Cambridge	MA

Given a base IRI *http://foo.example/DB/*, the direct mapping of this database produces a direct graph:

```
    @base <http://foo.example/DB/> .
```

[1] http://www.w3.org/2001/sw/rdb2rdf/

[2] http://www.w3.org/TR/r2rml/

[3] http://www.w3.org/TR/rdb-direct-mapping/

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<People/ID=7> rdf:type <People> .
<People/ID=7> <People#ID> 7 .
<People/ID=7> <People#fname> "Bob" .
<People/ID=7> <People#addr> 18 .
<People/ID=7> <People#ref-addr> <Addresses/ID=18> .
<People/ID=8> rdf:type <People> .
<People/ID=8> <People#ID> 8 .
<People/ID=8> <People#fname> "Sue" .

<Addresses/ID=18> rdf:type <Addresses> .
<Addresses/ID=18> <Addresses#ID> 18 .
<Addresses/ID=18> <Addresses#city> "Cambridge" .
<Addresses/ID=18> <Addresses#state> "MA" .
```

In this expression, each row, e.g. (7, "Bob", 18), produces a set of triples[4] with a common subject. The subject is an IRI formed from the concatenation of the base IRI, table name (People), primary key column name (ID) and primary key value (7). The predicate for each column is an IRI formed from the concatenation of the base IRI, table name and the column name. The values are RDF literals formed from the lexical form of the column value. In the example each foreign key produces a triple with a predicate composed from the foreign key column names, the referenced table, and the referenced column names. The object of these triples is the row identifier (<Addresses/ID=18>) for the referenced triple. Note that these reference row identifiers must coincide with the subject used for the triples generated from the referenced row. The direct mapping does not generate triples for NULL values.

2.2 R2RML

R2RML is a language for expressing customized mappings from relational databases to RDF datasets. Such mappings provide the ability to view existing relational data in the RDF data model, expressed in a structure and target vocabulary of the mapping author's choice. With R2RML a mapping author can define highly customized views over the relational data.

Every R2RML mapping is tailored to a specific database schema and target vocabulary. The input to an R2RML mapping is a relational database that conforms to that schema. The output is an RDF dataset[5], as defined in SPARQL, which uses predicates and types from the target vocabulary. The mapping is conceptual; R2RML processors are free to materialize the output data, or to offer virtual access through an interface that queries the underlying database, or to offer any other means of providing access to the output RDF dataset.

R2RML mappings are themselves expressed as RDF graphs and written down in Turtle syntax[6].

2.2.1 Mapping Overview

An R2RML mapping refers to logical tables to retrieve data from the input database. A logical table can be one of the following:

[4] A triple is composed by a subject, a predicate and an object.

[5] http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/#rdfDataset

[6] http://www.w3.org/TR/turtle/

- A base table;
- A view; or
- A valid SQL query (called an *R2RML view* because it emulates a SQL view without modifying the database).

Each logical table is mapped to RDF using a *triples map* (Figure 2). The triples map is a rule that maps each row in the logical table to a number of RDF triples. The rule has two main parts:

1. A *subject map* that generates the subject of all RDF triples that will be generated from a logical table row. The subjects often are IRIs that are generated from the primary key column(s) of the table.
2. Multiple predicate-object maps that consist of predicate maps and object maps (or referencing object maps).

Triples are produced by combining the subject map with a predicate map and object map, and applying these three to each logical table row. For example, the complete rule for generating a set of triples might be:

- **Subjects:** A template *http://data.example.com/employee/{empno}* is used to generate subject IRIs from the *empno* column.
- **Predicates:** The constant vocabulary IRI *ex:name* is used.
- **Objects:** The value of the ename column is used to produce an RDF literal.

By default, all RDF triples are in the *default graph* of the output dataset. A triples map can contain *graph maps* that place some or all of the triples into named graphs instead.

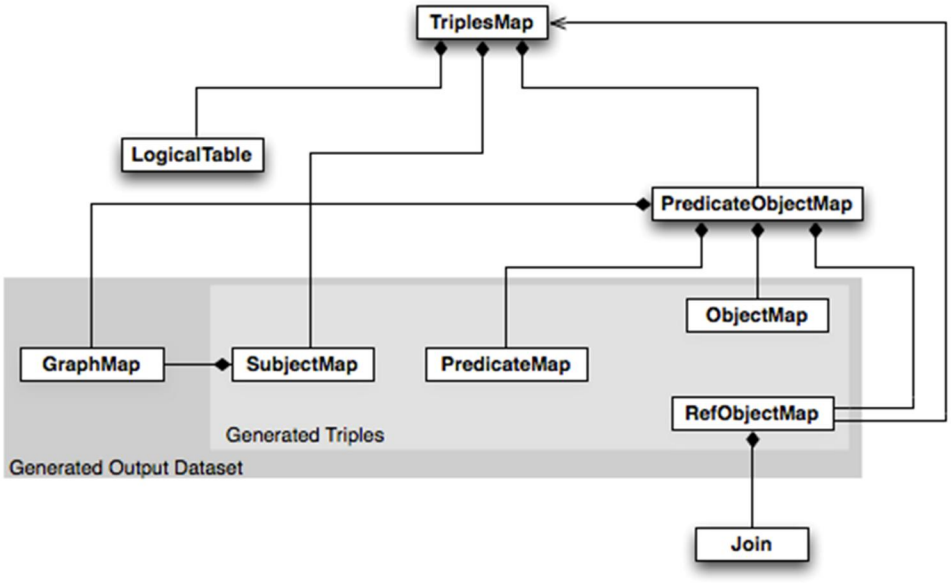

Figure 2. R2RML overview diagram

2.2.2 Examples

The following example database will be used in the following simplified examples. It consists of two tables, EMP and DEPT, with one row each:

EMP

EMPNO INTEGER PRIMARY KEY	ENAME VARCHAR(100)	JOB VARCHAR(20)	DEPTNO INTEGER REFERENCES DEPT (DEPTNO)
7369	SMITH	CLERK	10

DEPT

DEPTNO INTEGER PRIMARY KEY	DNAME VARCHAR(30)	LOC VARCHAR(100)
10	APPSERVER	NEW YORK

Extracting Records From a Single Table

The following R2RML mapping document will produce triples describing employees from the ***EMP*** table. The example below covers a subset of the table fields:

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix ex: <http://example.com/ns#>.
<#TriplesMap1>
    rr:logicalTable [ rr:tableName "EMP" ];
    rr:subjectMap [
        rr:template "http://data.example.com/employee/{EMPNO}";
        rr:class ex:Employee;
];
rr: predicateObjectMap [
        rr:predicate ex:name;
        rr:objectMap [ rr:column "ENAME" ];
].
```

The resulting RDF contains two statements: one introducing a single entity of type *ex:Employee* and a second statement assigning it a name property.

```
<http://data.example.com/employee/7369> rdf:type ex:Employee.
<http://data.example.com/employee/7369> ex:name "SMITH".
```

Computing a Property with an R2RML View

The following example defines a R2RML View (outside the database) pre-calculating certain properties (*STAFF*) and then refers this view from a *triples map*:

```
<#DeptTableView> rr:sqlQuery """
SELECT DEPTNO,
       DNAME,
       LOC,
       (SELECT COUNT(*) FROM EMP WHERE EMP.DEPTNO=DEPT.DEPTNO) AS STAFF
FROM DEPT;
""".

<#TriplesMap2>
    rr:logicalTable <#DeptTableView>;
    rr:subjectMap [
        rr:template "http://data.example.com/department/{DEPTNO}";
```

```
        rr:class ex:Department;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:name;
        rr:objectMap [ rr:column "DNAME" ];
    ];
    rr:predicateObjectMap [
        rr:predicate ex:location;
        rr:objectMap [ rr:column "LOC" ];
    ];
    rr:predicateObjectMap [
        rr:predicate ex:staff;
        rr:objectMap [ rr:column "STAFF" ];
    ].
```

The result RDF (note the *staff* property computed in the view):

```
<http://data.example.com/department/10> rdf:type ex:Department.
<http://data.example.com/department/10> ex:name "APPSERVER".
<http://data.example.com/department/10> ex:location "NEW YORK".
<http://data.example.com/department/10> ex:staff 1.
```

Linking Two Tables

The following fragment builds a bridge between the two tables expressed as an object property in RDF:

```
<#TriplesMap1>
    rr:predicateObjectMap [
        rr:predicate ex:department;
        rr:objectMap [
            rr:parentTriplesMap <#TriplesMap2>;
            rr:joinCondition [
                rr:child "DEPTNO";
                rr:parent "DEPTNO";
            ];
        ];
    ].
```

It performs a join between the *EMP* table and the R2RML view defined in the previous example, on the *DEPTNO* columns. The objects will be generated from the subject map of the parent triples map, yielding the desired triple:

```
<http://data.example.com/employee/7369> ex:department
<http://data.example.com/department/10>.
```

3 RDB2RDF Implementations

This section provides an update on the current implementations of the *W3C RDB2RDF* standards. It does not cover all the tools listed as accepted RDB2RDF implementations[7] due to various reasons – insufficient information, limited reusability, etc.

3.1 db2triples (update)

The *db2triples*[8] implements the latest versions of both Direct Mapping and R2RML. It was introduced in the previous version of this document (D.2.2.1). However, it was neither aligned with the latest by versions of the DM and R2RML at that time, nor was it stable enough to be used in CUBIST. The latest release of db2triples shows significant improvement and it was successfully deployed for the ETL in one of the use cases. Details on the actual ETL process can be found in section 4.3.

In a nutshell, db2triples accepts as input RDBMS connection parameters and a R2RML transformation document (and no mapping in case of *Direct Mapping* mode). The output is the RDF dump of the database data based on the transformation mode used. The tool manages the memory efficiently, which in turn enables it to process large amounts of data.

Supported RDBMS: MySQL, PostgreSQL

License: LGPL

3.2 Ultrawrap

Ultrawrap[9] is a RDB2RDF wrapper system that automatically directly maps relational database schema into an OWL ontology and exposes the relational database contents as RDF and through a SPARQL endpoint using the W3C Direct Mapping. Additionally, users can manually create custom mappings using the R2RML mapping language.

Ultrawrap's unique architecture enables SPARQL queries to be optimized by the SQL engine. Therefore, a SPARQL query's execution time is comparable to its semantically equivalent SQL queries execution time.

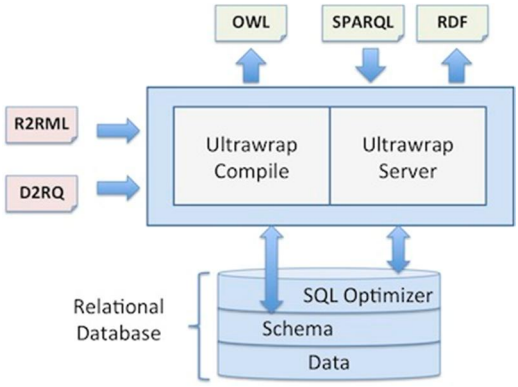

Figure 3. Ultrawrap's architecture scheme.

[7] http://www.w3.org/2001/sw/rdb2rdf/wiki/Implementations

[8] https://github.com/antidot/db2triples/

[9] http://www.capsenta.com/product.html

Ultrawrap (Figure 3) comprises of two components: Ultrawrap Compile and Ultrawrap Server. Ultrawrap Compile only runs at the beginning of the execution and then Ultrawrap Server continues and sets up the SPARQL endpoint on a Jetty Server.

The following diagram (Figure 4) illustrates the workflow of these two components:

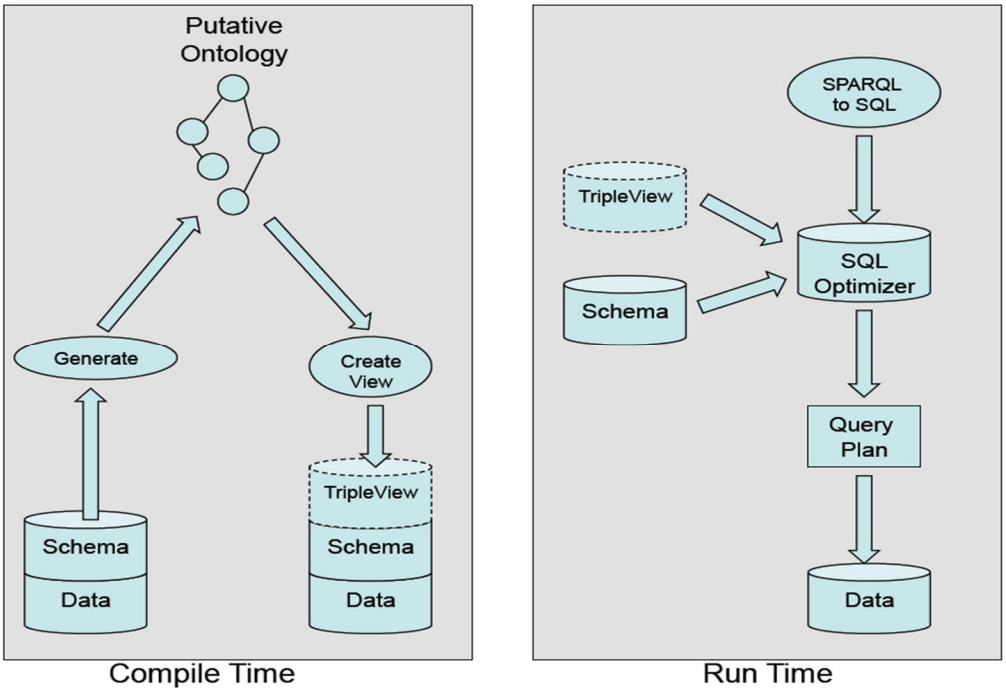

Figure 4. The workflow of Ultrawrap Compile and Ultrawrap Server.

At compile time, an OWL ontology is extracted based on the database schema definition, along with a Direct Mapping expressed in R2RML. This mapping serves as a backbone for customized RDF graph mappings. On the base of this mapping, a *triple view* is generated to display a logical RDF representation of the relational data.

At runtime, the incoming SPARQL requests are rewritten in SQL without any optimisation. It is responsibility of the specific RDBMS optimiser to build optimal query plan for the SQL queries.

Supported RDBMS: Oracle, IBM DB2, Microsoft SQL Server, PostgreSQL

License: commercial

3.3 D2RQ Platform (update)

This tool was described in the previous version of this document (D.2.2.1) as one of the candidates to transform relational data into RDF triples. Some updates regarding the W3C specification advances:

- Direct Mapping support – Announced as experimental feature, Direct Mapping can now be used instead of the native D2RQ mapping.
- R2RML – It is still not supported in the current version, although it was initially scheduled for Q2 of 2012.
- SPARQL 1.1 – Preliminary support announced in version 0.8 (12 March, 2012).

Supported RDBMS: MySQL, PostgreSQL, MS SQL Server, Oracle, HSQLDB

License: Apache License v.2.0

4 Semantic ETL from Structured Data Sources in CUBIST

The following Section gives an update on the semantic ETL from structured data sources in CUBIST. Deliverable D2.2.1 and the previous sections analysed the state of the art of existing tools for this task. Furthermore, D2.3.1 already discussed preliminary semantic ETL processes for the three use cases. We will now recall the structured data sources for each use case and discuss the proper tools. The final Semantic ETL process for each use case will be discussed in deliverable D2.3.2.

4.1 WP7 HWU

The structured data for this use case mainly exists in the two databases EMAP and EMAGE being described in detail in the *Data and interface design document* D7.2.1. The data is complemented by a comma-separated values file containing *Theiler stage* information.

Due to the involvement of different types of structured data and the request to map this data, we identified *Talend Open Studio*[10] for Data Integration as the best candidate tool for the semantic ETL process in CUBIST. On the one hand, it offers a great variety of components to integrate all popular types of RDBMS, different types of local files (CVS, XML, and TXT) and different kinds of data processing (aggregation, filtering, sorting, mapping, etc.). On the other hand, it provides additional RDF components to parse and serialize RDF data. Its graphical user interface makes it easy to use and the application of contexts allows an easy adaption of the ETL process to different environments. Moreover, the viral GPL license does not imply any restrictions on the usage and the outcomes of the tool.

4.1.1 Summary of ETL process

The ETL process is split into sub-jobs responsible for specific input data (*Gene*, *Theiler Stage*, *Tissue*, *Textual Annotations* and *Experiment* data). Each sub-job consists of the following major processing steps:

1. Fetching the data from a database or a file;
2. Cleaning and Normalizing the input data;
3. Generating the RDF graph structure;
4. Serialization to a set of files.

The first step is relatively trivial in case of fetching data from a tabular file, as it only covers reading the file with a suitable Talend Open Studio component and passing the data through. When data is retrieved from the databases EMAGE and EMAP it is filtered with the help of corresponding SQL queries and then passed to the next step. Step 2 covers cleaning the data - especially removing whitespaces and special characters (e.g., dots) that cause problems when directly transforming to the triple data. Furthermore, uniform identifiers (URI) are generated. In step 3 triple data is generated according to the classes and properties defined in the ontology and based on the input data. Finally, in step 4 all generated triples are written into temporary files in N-Triple format. This file format was chosen as it is eminently suitable for streaming the data. The temporary N-Triple files generated during the sub-jobs are collected and united in a post procedure (post-job) and written to one single output file to facilitate bulk loading of the data into the triple store (Figure 5).

[10] http://de.talend.com/products/talend-open-studio

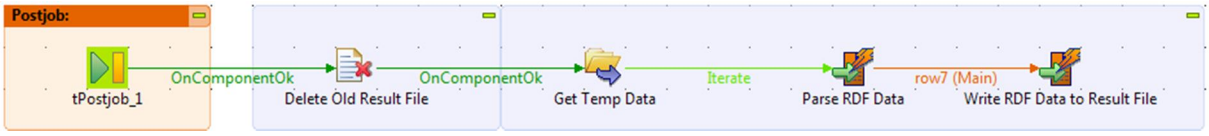

Figure 5. Talend Open Studio Post-job to summarize the output of the sub-jobs

An example of a sub-job – the one responsible for the transformation of *Tissue* data – is depicted in Figure 6. The processing steps 2 and 3 are executed in one step, as the data cleaning and normalization task is not too heavy. First the relevant data is read from the database. The retrieved data is then cleaned and mapped onto triples. Finally, the triples are written into temporary files.

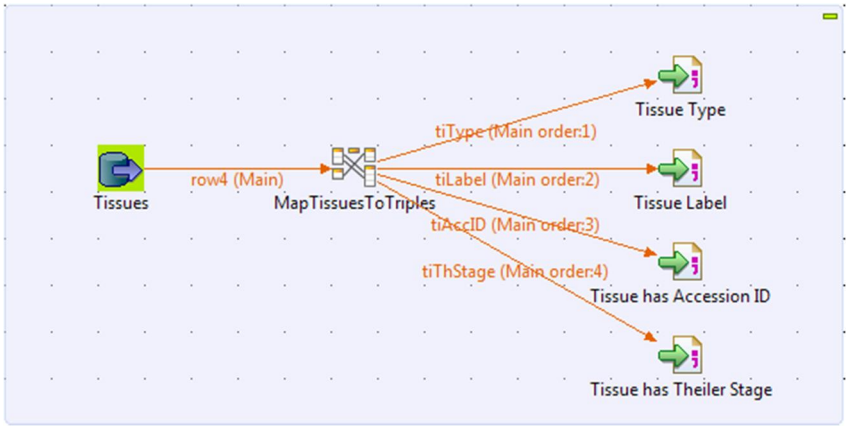

Figure 6. Talend Open Studio Sub-job for the transformation of tissue data.

4.2 WP8 SAS

The structured data provided by Space Applications Services is the CUBIST Space Data Pack – a TSV file including 30 days of pre-processed telemetry data. Telemetry measurements are taken each second for more than 200 parameters. From this it follows that a huge amount of data has to be transformed and more than 200 parameters have to be described/named for the transformation process. As a manual handling of this task is impractical and error-prone Talend Open studio for Data Integration was again chosen as the proper tool for the semantic ETL process. As already mentioned, it offers a great variety of components and it is easy to use. It provides a programming interface to automate the description of the more than 200 parameters. Furthermore, it scales well to large amounts of data.

4.2.1 Summary of ETL process

The transformation expects as input a TSV file or a set of TSV files (if the input is split into smaller pieces). The result is one or more RDF files corresponding to the input files. The representation format of the resulting RDF is TURTLE which is one of the most efficient storage formats and has the advantage of being relatively human readable.

To provide further readability and data storage, the transformation process uses namespaces (mdb, xsd, rdf) to abbreviate repeating URIs. The ETL job – depicted in Figure 7 consists of a pre-job and the following main transformation. The pre-job takes the input files (*TSVFiles*) and produces for each a result file containing the namespaces definition only (component *CreateFilesAndNamespaces*). The

new file names are derived from the corresponding original ones (004-ready.dump -> 004-ready.dump.n3).

The main transformation then reopens each of these files, generates the RDF triples (using the namespaces definitions) and prints the generated data into files. The RDF triples are generated as follows by iterating the elements of each row from the input files:

1. Detecting the timestamp property (column 'Time') and producing the corresponding triples in RDF. The type of the time literal (xsd:dateTime) is explicitly stated.
2. For the remaining parameters:
    a. Constructing the corresponding predicate URI from the parameter name,
    b. Determining the data type (integer, float, string) from the parameter value and constructing the RDF value with the type explicitly stated (xsd:integer, xsd:float, xsd:string),
    c. Producing the triple: <packet-uri> <parameter-uri> <data value>.

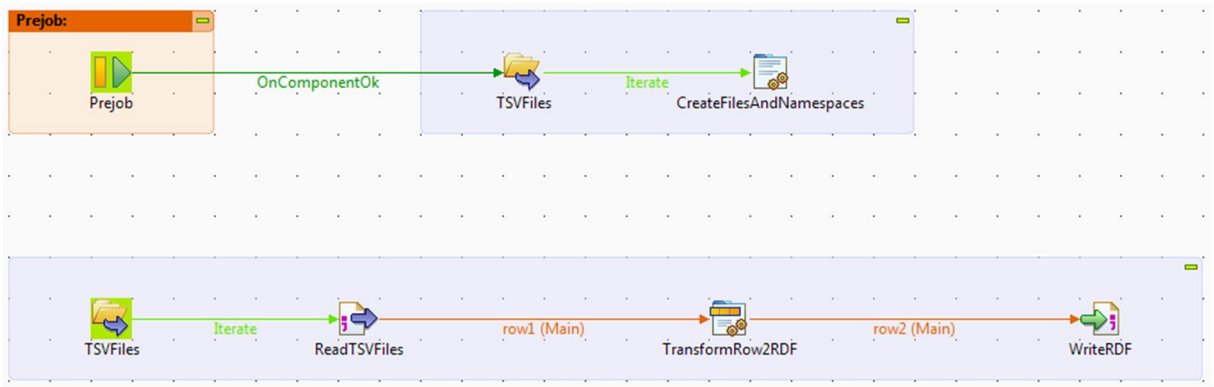

Figure 7. Talend Open Studio Job for SAS use case

Figure 8 shows a fragment of the generated triple data.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix mdb: <http://www.spaceapplications.com/owl/2011/08/cubist/mdb#> .

mdb:packet_2008-09-23T09_47_03 a mdb:Packet;
mdb:hasTime "2008-09-23T09:47:03"^^xsd:dateTime;
mdb:has_PAR--Grd--SOLAR_PB1_Ovtemp "NO"^^xsd:string;
mdb:has_PAR--Grd--SOLAR_PB3_Outlet1_Trip_Stat "OFF"^^xsd:string;
mdb:has_PAR--Grd--SOLAR_PB3_Outlet2_Trip_Stat "OFF"^^xsd:string;
mdb:has_PAR--Grd--SOLAR_PB3_Outlet3_Trip_Stat "OFF"^^xsd:string;
mdb:has_PAR--Grd--SOLAR_PB3_Outlet4_Trip_Stat "OFF"^^xsd:string;
mdb:has_PAR--Grd--SOLAR_Ancillary_Stat "OK"^^xsd:string;
mdb:has_PAR--Grd--SOLAR_PB1_Stat "OK"^^xsd:string;
mdb:has_PAR--Grd--SOLAR_PB1_DC_DC_Converter_Stat "ON"^^xsd:string;
mdb:has_PAR--Grd--SOLAR_PB1_5V_Service_Stat "ON"^^xsd:string;
........
mdb:has_PAR--Grd--SOLAR_PB1_5V_VME_Outlet_Current "4.325"^^xsd:float;
mdb:has_PAR--Grd--SOLAR_PB1_12V_Service_Outlet_Voltage "12.093"^^xsd:float.
```

Figure 8. Triple data for SAS use case (fragment)

To run the Talend job for the Space Applications use case two parameters should be provided as context values for the job: input_dir and result_dir.

4.3 WP9 INN

The structured data of the Innovantage use case is delivered as MySQL database dumps. It is the only data that has to be converted to RDF and R2RML mappings are the proper techniques for the Semantic ETL process. The tool of choice applied therefore is *db2triples* from Antidot (see Section 3.1). It has passed the official R2RML compliance tests and is thus a *W3C-validated implementation*. Moreover, the LGPL license does not imply any restrictions on the usage and the outcomes of the tool for CUBIST.

4.3.1 Summary of ETL process

The tool db2triples was successfully used to convert the Innovantage use case data from the relational MySQL representation to RDF data (triples dump). The conversion used entirely R2RML mapping.

The mapping is split into multiple mappings that divide the data into logical blocks:

- jadvertiser.n3: Mapping of *advertisers* and their links to other data.
- jcategory.n3: Mapping of *disciplines and sub-disciplines*.
- jcontact.n3: Mapping of *contacts*.
- jjobboard.n3: Mapping of *job boards*.
- jlocation.n3: Mapping of *locations*.
- jml.n3: Mapping of *company profiles* (presently unused as company data is not available to the project due to licensing issues).
- jvacancy.n3: Mapping of *vacancies* and their links to other data.

Each of the above mappings has to be run independently with db2triples. It does not matter in what order the various mappings are processed. Db2triples is invoked on the command line (presuming the class path was set properly):

```
java net.antidot.semantic.rdf.rdb2rdf.main.Db2triples -m r2rml -b
<db> -u <user>\

-p <password> -t N3 -r <input-name> -o <output-name>
```

Where <db>, <user> and <password> are respectively the database name, user and password, while <input-name> and <output-name> are respectively the input R2RML mapping filename (e.g. jadvertiser.n3) and a convenient output filename where the converted RDF data will be written (e.g. Advertiser.n3).

Once all of the mappings have been processed, the resultant RDF files (one for each input mapping) can be used for further processing where RDF data is needed, e.g. populating an RDF store.

5 Conclusion

This deliverable provides the second version of the analysis of CUBIST use case requirements for extracting information from structured data sources, an update of existing solutions to support these requirements, as well as an update on the Semantic ETL processes within the three use cases. The changes since D2.2.1 focus on updates on RDB2RDF techniques especially in context with *W3C* recommendations. This document identifies the most appropriate tools for the semantic ETL processes for each of the three use cases. A more detailed technical description of the semantic ETL processes along with the upload process will be included in the second integrated prototype of the CUBIST Data Integration and Federation Platform (D2.3.2).