Collaborative Project

GeoKnow - Making the Web an Exploratory Place for Geospatial Knowledge

Project Number: 318159 **Start Date of Project:** 2012/12/01 **Duration:** 36 months

Deliverable 6.1.2

Report on Customer Data Preparation and Transformation for Linked Data Usage

Dissemination Level	Public
Due Date of Deliverable	M12, 30/11/2013
Actual Submission Date	M12, 30/11/2013
Work Package	WP 6, GeoKnow for E-Commerce
Task	T6.1
Type	Report
Approval Status	Approved
Version	1.0
Number of Pages	17
Filename	D6.1.2_Customer_data_preparation.pdf

Abstract: This report provides a description of the preparation and transformation the data sets described in report D6.1.1, along with potential integration points between the selected data and geospatial linked open data provided by WP2 and WP3.

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

History

Version	Date	Reason	Revised by
0.1	21/10/2013	Initial version of this deliverable	Matthias Wauer
0.2	23/10/2013	CRM data transformation description	Matthias Wauer
0.3	25/10/2013	Additional data transformation	Matthias Wauer
0.4	20/11/2013	Sparqlify section	Claus Stadler
0.5	20/11/2013	Internal review, additions	Andreas Both
0.6	25/11/2013	Consolidated review version	Matthias Wauer
0.7	27/11/2013	Peer review	Robert Isele
0.9	28/11/2013	Final changes w.r.t. review comments	Matthias Wauer
1.0	29/11/2013	Final version	Andreas Both

Author List

Organization	Name	Contact Information
Unister	Matthias Wauer	matthias.wauer@unister.de
Unister	Andreas Both	andreas.both@unister.de
InfAI	Claus Stadler	cstadler@informatik.uni-leipzig.de
BROX	Robert Isele	robertisele@gmail.com

Executive Summary

Linking internal data with the geospatial open Linked Data created in the course of GeoKnow is one of the major opportunities for companies arising from this project. Representing an e-commerce use case, Unister is providing a number of anonymised internal data sets from a travel/tourism context to demonstrate the added benefits of their integration. This deliverable describes how these data sets are transformed into Linked Data. It also discusses the methods and vocabularies used in this process, as well as additional data sets which can be integrated.

Table of Contents

1 Introduction

Unister is one of Europe's leading e-commerce providers with access to customer data of millions of bookings, flights and other information. Two available data sets, as summarized in Table 1, have been selected for use within the GeoKnow project, which are described in more detail in deliverable report D6.1.1.

	flight booking	hotel reservation
content	flight booking information from several Unister portals	user-based hotel reviews and recommendations
number of records	5 216 357	1 162 822
first entry	12/02/2007	29/01/2002
last entry	06/06/2013	15/06/2013

Table 1: Summary of the D6.1.1 data sets

Naturally, when speaking about personal customer data, their anonymised version is meant. Before these data sets can be used in GeoKnow, the data has to be accessible, i.e., available for GeoKnow applications as Linked Data. This report covers the following aspects:

- Different approaches on how the data can be integrated in general
- Steps required for providing the data, such as extraction and transformations
- Mappings to existing vocabularies and ontologies

In addition to that, Section 7 will address additional data sets. Those not publicly available were not provided to GeoKnow partners due to licensing constraints, but they are valuable for the "GeoKnow for E-Commerce" scenario of work package 6. Here, we describe which extraction and transformation methods we have applied in order to process these sources as Linked Data.

2 Data Integration Approaches

In general, there are two ways to access disparate data. The data sets can be transferred into a central target datastore, with optional processing on the fly or in the target datastore to conform to a certain schema. This approach is typically called an *ETL* (extract, transform, load) process. The original data has to be extracted from the data source, then the data can be transformed into a target schema. The transformed data can then be stored into the target datastore.

Another option is *data federation*: client requests for certain data are translated by an adapter into a request for an endpoint on the data source. The provided data is again translated by the adapter and returned to the client. Thus, the information does not have to be stored centrally, but this approach requires query processing capabilities on the original data source. In addition, the client or a middleware have to support such federated requests.

Both approaches can have very different key characteristics. In theory, for the Unister data sets 1 and 2 "Flight Bookings" and "Hotel Reservations", both approaches would be feasible given that the original databases can be accessed. For the third data set "Facebook", there is no final database schema yet, so a reasonable data integration approach can't be decided at this point.

In practice, some of the data fields require costly transformation in order to integrate them with other data sets. Examples include the address and country of residence in the Flight Bookings data set, and the hotel city information in the Hotel Reservations data set. These data fields require complicated mapping processes to other data sets. It is, however, feasible to defer such processing to a later stage. We will go into more detail with regards to this aspect in Section 4.

3 Extraction Methods

The extraction process is crucial concerning quality and quantity of available information. Therefore, we investigate methods and tools supporting the data extraction and compiling process.

The Unister data sets are stored in relational databases (MySQL), but we also have to integrate data sets available as XML feed with XML Schema definition (see Section 7). We have applied and tested the following extraction methods on the provided data sets:

- Spring Batch[1] is a Spring based framework for implementing ETL processes.
- Scriptella[2] is a more lightweight alternative to SpringBatch.
- D2RQ[3] is a method for providing access to relational databases via a virtual SPARQL endpoint. The required transformation mapping is expressed in RDF itself, using a D2RQ-specific vocabulary.
- Sparqlify[4] is similar to D2RQ in that it provides a SPARQL endpoint to relational database content. As such, it also includes the transformation into RDF using the Sparqlification Mapping Language (SML)[5].

There are several tools similar to Sparqlify and D2RQ[6]. As a consequence, the World Wide Web Consortium (W3C) working group RDB2RDF published a recommendation[7] for a standardised mapping vocabulary. D2RQ already supports this standard, support in Sparqlify is planned.

3.1 Spring Batch

Spring Batch does not focus on RDB to RDF mappings, but provides a generic framework for implementing batch processes. It defines the notions of a job, job steps, item readers, item processors, and item writers, which can be used for building ETL processes. As such, it comes with many components for extracting data from several types of data sources and managing an ETL workflow, but it falls short of providing any RDF transformations out of the box. It is, however, possible to implement so-called RowMappers which transform a result set row from a relational database into RDF.

We used this framework to implement an ETL process for loading internal hotel information from XML dumps and other relational databases. Spring Batch helped here by providing an environment for combining modules (job steps) for pre-processing data, extracting multiple input files, and mapping the input data into data transfer objects. Additionally, it is relatively easy to enable parallel extraction of this data, using so-called partitions, which can massively reduce processing times[8]. In the following Listing 1, parts of the Spring Batch configuration of an import process is shown.

[1] http://projects.spring.io/spring-batch/
[2] http://scriptella.javaforge.com/
[3] http://d2rq.org/
[4] http://sparqlify.org/
[5] http://sparqlify.org/wiki/SML
[6] for a list of such tools, refer to http://www.w3.org/2001/sw/rdb2rdf/wiki/Implementations
[7] http://www.w3.org/TR/r2rml/
[8] Depending on the processing task, we experienced best results using n=8 to n=16 partitions on a quad-core server when running jobs sequentially. However, the partitioning can introduce some initial delay by posing n simultaneous queries with OFFSET and LIMIT clauses at the data store. Since this often appears to be the limiting factor, we are going to work on a more efficient locally caching partitioning, which will be slightly less transparent

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" ...>

  <!-- JOB: Importing Hotel Data -->
  <batch:job id="importHotelsJob" parent="abstractJob" restartable="true"
     >
    <batch:listeners merge="true">
      <batch:listener ref="hotelsQualityMetricsListener" />
    </batch:listeners>
    ...
    <batch:step id="importHotels" parent="abstractStep" next="
       importHotelsPostProcessingStep">
      <batch:tasklet>
        <batch:chunk reader="hotelResourceItemReader" processor="
           hotelCompositeItemProcessor"
          writer="hotelCompositeItemWriter" commit-interval="\${hotel.
             batchSize}" skip-limit="\${hotel.skipLimit}"
          retry-limit="\${common.retryLimit}">
          <batch:retryable-exception-classes>
            <batch:include class="java.lang.Exception" />
          </batch:retryable-exception-classes>
          <batch:skippable-exception-classes>
            <batch:include class="java.lang.Exception" />
          </batch:skippable-exception-classes>
        </batch:chunk>
      </batch:tasklet>
    </batch:step>
    ...
  </batch:job>

  <bean id="hotelResourceItemReader" class="org.springframework.batch.
     item.xml.StaxEventItemReader">
    <property name="resource" value="\${hotel.source.file}" />
    <property name="fragmentRootElementName" value="Hotel" />
    <property name="unmarshaller">
      <bean class="org.springframework.oxm.jaxb.Jaxb2Marshaller">
        <property name="classesToBeBound">
          <array>
            <value>com.unister.geoknow.import.JaxbHotel</value>
          </array>
        </property>
      </bean>
    </property>
    <property name="maxItemCount" value="\${hotel.xml.maxItemCount}" />
  </bean>
  ...
```

Listing 1: Spring Batch import job configuration example

The XML markup introduces some configuration overhead, but also enables a highly modularised use of the implemented components. For example, we combined a component reading from multiple ZIP resources with another for extracting them and processing each line with a custom row mapper, which created RDF statements from this data.

The interfaces of Spring Batch cause a few issues, especially when parallelising jobs with partitions. The read method (as well as the process and write methods) has no context parameter, so it is not possible to find out (and, hence, to log) which partition is handling an item. The step and chunk contexts don't help either, since the partitioning works with a single ItemReader instance instead of assigning a new instance to each partition. As a result, using listeners for logging item handling does not work properly, for example

when it comes to logging the read duration.

3.2 Scriptella

Similar to Spring Batch, Scriptella uses a custom XML document type definition as a vocabulary to describe an ETL process. We used this tool for simple database transformation jobs and for extracting N-Triples[9] files from databases, which builds RDF statements from database entries. It is possible to process the data using JavaScript, but other than that the abstraction level is relatively low.

The following Listing 2 shows the entire Scriptella ETL process definition for transferring the hotel stars property from one database into another. All sources are configured as `connection` elements, the extraction is configured in `query` elements, and transformation or load tasks are configured in `script` elements.

```
<!DOCTYPE etl SYSTEM "http://scriptella.javaforge.com/dtd/etl.dtd">
<etl>
  <description>copies the stars data from Hotel table to HotelProperty
     table.</description>

  <!-- connection used for querying the hotel database -->
  <connection id="hotel" driver="auto" url="jdbc:mysql://testdb.intranet.
     lan/hotel?autoReconnect=true"
    user="hotel" password="<password>" lazy-init="true">
    autocommit=true
  </connection>
  <!-- scripting support (javascript) -->
  <connection id="js" driver="script" />

  <!-- create hotel property table if not exists -->
  <script connection-id="hotel">
    CREATE TABLE IF NOT EXISTS `hotel`.`HotelProperty` (
    ...
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8
  </script>

  <!-- initialise a counter and start -->
  <query connection-id="js">
    java.lang.System.setProperty('count', 1);
    query.next();

    <query connection-id="hotel">
      SELECT Hotel.id, Hotel.stars FROM Hotel WHERE Hotel.stars IS NOT
         NULL

      <query connection-id="hotel">
        (
          SELECT om.uri as uri
          FROM OntologyMapping om
          LEFT JOIN HotelHistory hh
          ON hh.previous = om.hotel_id
          WHERE hh.hotel_id = ?id
        )
        UNION
        (
          SELECT distinct om.uri as uri
          FROM OntologyMapping om
```

[9] http://www.w3.org/TR/n-triples/

```
        WHERE om.hotel_id = ?id
      )

      <!-- logging -->
      <script connection-id="js">
        java.lang.System.out.println("Writing stars data " + java.lang.
           System.getProperty('count') + ": " + uri + " " + stars);
        java.lang.System.setProperty('count', parseInt(java.lang.System
           .getProperty('count')) + 1);
      </script>

      <!-- insert data into hotel property table -->
      <script connection-id="hotel">
        REPLACE INTO HotelProperty (hotel_uri, property_uri, object_uri
           , value)
        VALUES
        ('\$uri', 'http://ontology.unister.geoknow.eu/ontology#stars',
           'http://www.w3.org/2002/07/owl#Nothing', '\$stars');
      </script>

    </query>
  </query>
</query>
</etl>
```

Listing 2: Scriptella ETL process example

We only applied Scriptella for rather straightforward tasks because it is much less modular than Spring Batch, but still quite useful in such scenarios.

3.3 D2RQ

D2RQ is a set of tools which enable RDF views on relational databases. We applied it on two of our source databases: a GIATA database containing city and weather data, and an internal geoinformation database mostly consisting of information on cities and regions. The included tool `generate-mapping` was used to build an initial automated mapping. The resulting turtle mapping file was then adapted in order to build custom URI patterns and define additional conditions not explicitly defined in the database schema. The following Listing 3 shows parts of an internal geoinformation database mapping.

```
@prefix map: <#> .
@prefix db: <http://ontology.unister.geoknow.eu/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#> .
@prefix jdbc: <http://d2rq.org/terms/jdbc/> .
@prefix d2r: <http://sites.wiwiss.fu-berlin.de/suhl/bizer/d2r-server/
   config.rdf#> .
...

<> a d2r:Server;
  rdfs:label "Geo DB Mapping Server";
  d2r:sparqlTimeout 60;
  ...
  .

map:database a d2rq:Database;
  d2rq:jdbcDriver "com.mysql.jdbc.Driver";
```

```
  d2rq:jdbcDSN "jdbc:mysql://testdb.intranet.lan/geo";
  d2rq:username "geo";
  d2rq:password "<password>";
  jdbc:autoReconnect "true";
  jdbc:zeroDateTimeBehavior "convertToNull";
  jdbc:keepAlive "3600";
  d2rq:resultSizeLimit "10000";
  d2rq:fetchSize "10000";
  .


# GeoID
map:geoID a d2rq:ClassMap;
  d2rq:dataStorage map:database;
  d2rq:uriPattern "urn:geo-database-v1-id-@@semweb_geo.geo_id@@";
  d2rq:class gdb:Place;
  d2rq:classDefinitionLabel "GeoID";
  d2rq:condition "semweb_geo.geo_id <> 0";
  d2rq:condition "semweb_geo.source_geonames = 'Y'";
  .

# hierarchy_type
map:geoID_hierarchyType a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:geoID;
  d2rq:property rdf:type;
  d2rq:propertyDefinitionLabel "hierarchyType mapped to rdf:type";
  d2rq:uriPattern "urn:geo-database-v1-hierarchyType-@@semweb_geo.
     hierarchy_type@@";
  .

...
```

Listing 3: D2RQ mapping example

We experienced a rather straightforward mapping and usable performance for relatively simple queries. However, even with specifying the JDBC connection parameters as shown above we experienced that D2RQ did not return or refresh an idle database connection, but reused the timed out connection after some time, so it was necessary to implement retries of such failed requests.

3.4 Sparqlify

Sparqlify is the name of a SPARQL-to-SQL rewriting project aimed at providing virtual SPARQL access to the full OpenStreetMap data set in the course of the LinkedGeoData[10] (LGD) project. For this purpose, D2RQ turned out to be unsuitable due to several restrictions, such as scalability issues and the lack of spatial support. In GeoKnow, work on both Sparqlify and LinkedGeoData continues. With Sparqlify, RDB-to-RDF mappings are expressed using the *Sparqlification Mapping Language*[11] (SML). An excerpt of a manually crafted mapping for RDFising a database containing hotels and corresponding reviews is shown in Listing 4. The data was originally stored in a MySQL database, which Sparqlify does not support yet. Fortunately, for the sake of evaluation, the ruby package *mysql2psql* could be successfully applied for converting the database to PostgreSQL.

```
Prefix xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix owl: <http://www.w3.org/2002/07/owl#>
Prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
```

[10] /urlhttp://linkedgeodata.org
[11] http://sparqlify.org/wiki/SML

```
Prefix acco: <http://purl.org/acco/ns#>

Prefix r: <http://geoknow.eu/resource/>
Prefix o: <http://geoknow.eu/ontology/>

Create View hotels As
  Construct {
    ?s
      a acco:Hotel ;
      o:id ?id ;
      rdfs:label ?l ;
      acco:feature ?sf ;
      geo:lon ?lon ;
      geo:lat ?lat ;
      vcard:hasAddress ?a ;
      .
  }
  With
    // Note: Variables on the
    // right-hand-side refer to table columns, whereas those on the
    // left-hand-side are used to instantiate the Construct-template
    ?s = uri(r:, 'hotel', ?id)
    ?id = typedLiteral(?id, xsd:int)
    ?l = plainLiteral(?name)
    ?sf = uri(r:, 'hotel-rating', ?id)
    ?lon = typedLiteral(?lon, xsd:double)
    ?lat = typedLiteral(?lat, xsd:double)
    ?a = uri(r:, 'hotel-address', ?id)
  From
    geoknow_hotels
```

Listing 4: SML mapping example

3.5 Summary

The presented extraction methods differ in how they can and should be applied. The ETL oriented methods, particularly the more modular Spring Batch, are reasonable for transforming entire data sets and support much more complex transformations. The mapping-based data source wrappers like Sparqlify, on the other hand, are very well suited for rapid integration and ad-hoc mashups of different sources. It is, however, also possible to combine both methods, i.e., to use a Spring Batch job for loading a data set transformed using a D2RQ or Sparqlify mapping. A similar combination can also be applied for interlinking or enrichment transformations, as explained below.

4 Data Transformation

For the RDB to RDF extraction tools explained above, the transformation of relational database schemas into the RDF data model is already executed in the extraction phase. Still, the original database fields contain information that has to be further transformed in order to create final representation. For example, some hotel reviews are badly encoded and should be converted accordingly, and geocoordinates encoded as GeoRSS point[12] may need to be available as separate latitude and longitude statements. Specific required transformations are discussed below, but in general, data transformations can be required in the following cases:

- transforming IRIs into other IRI, e.g., into another namespace (IRI transformation)
- transforming literal values into another literal or IRI (literal transformation)
- transforming a set of statements (either IRI or literal values) into another literal or IRI (multi-statement transformation)

We only apply very straightforward transformations during the extraction jobs, as shown in this excerpt of Listing 3:

```
...
map:geoID_hierarchyType a d2rq:PropertyBridge;
  d2rq:uriPattern "urn:geo-database-v1-hierarchyType-@@semweb_geo.
     hierarchy_type@@";
...
```

Here, a certain column of a database row is used as a placeholder in an IRI pattern (literal transformation). We decided to address more complex transformations later on in the process for two reasons. For one thing, the mapping languages are not capable of defining IRI mappings between ontology definitions, so if a target vocabulary describes hierarchy types in a different way than the original database, a mapping between source values and target IRIs can't be expressed in such a RDB to RDF mapping. For another thing, defining more complex multi-statement transformations with such IRI mappings is not possible with these mapping languages, it is simply out of their scope and should be done in a following schema mapping.

Therefore, we implemented subsequent transformation jobs using Spring Batch. Similar results might be achieved by creating and applying appropriate mappings for schema mapping tools, such as R2R[13]. We implemented a set of item processors on a custom item class `Instance`, which encapsulates an OpenRDF[14] Sesame RDF `Model`.

Initially, we experimented working with processes on statements (instead of working on instances, i.e., a single entity, or entire graphs), but soon found that this is too limited for more complex transformations, which depend on information stored in multiple statements. For example, states (administrative divisions) have an ISO identifier consisting of country and state code. In order to generate this code, we have to know not only the state code of the instance, but also the country it is located in and its country code. Therefore, we also introduced the concept of *multi-instance enrichment*, which provides the necessary information in a single item to the Spring Batch item processors. These item types are also indicated in Table 2, which describes some of the transformations we have implemented using such processors.

[12] http://georss.org/model.html
[13] http://r2r.wbsg.de/spec/, also supports custom transformation functions
[14] http://www.openrdf.org/

transformation	description	item type
Geocoordinates	Transforms different geocoordinate representations. Geocoordinates are typically represented by separate latitude and longitude statements, but can also be in GeoRSS point notation or in separate statements for each coordinate degree, minute, and second.	instance
Filtering	Filters certain statements, typically based on their predicate, object, and/or graph. Examples include filtering obviously erroneous numbers, e.g., for population or area of a place.	statement
Conversion	Converts certain statements, such as different area representations (square meters, square kilometers, square miles, etc.) into a unified representation.	statement
Literals	Translates or filters parts of a literal. For instance, some labels and descriptions in source data sets have partially invalid encodings or additional markup, such as Wikipedia infoboxes which have not been filtered correctly.	statement
Hotel	Custom transformations for certain instance classes, e.g., splitting the city information associated to a hotel instance into a separate country instance and adding a relationship "hotel located in city" to the hotel instance.	multi-instance
Interlinking	Adds additional statements, such as `owl:sameAs` or specific identifiers, to an instance via custom lookup processes. Some of these could also be applied using existing frameworks, such as SILK (see http://lod2.eu/Project/Silk.html).	instance

Table 2: Transformation steps

5 Data Load

When the extracted data has been transformed, it can be loaded into a data sink, i.e., the target database. In our case, we wrote the resulting RDF (the statements of the Spring Batch item) into a Virtuoso store, using a Spring `JdbcTemplate` and Virtuoso's `DB.DBA.TTLP` function.

We experienced increasing memory load when improving the performance of the transformation and load processes by partitioning, i.e., parallel computation. We were able to track this down to Virtuoso's JDBC[15] driver, which puts a high load on the JVM[16] finalizer. The JVM just creates one finalizer for removing garbage collected instances of, e.g., result sets and rows. This finalizer can't keep up with the high amount of such objects being created and removed in parallel transformation jobs with a high I/O ratio. The only option for us was to circumvent this issue was manually invoking the JVM's finalizer on each Spring Batch read operation, as shown in Listing 5:

```
public synchronized T read() throws Exception {
  System.runFinalization();
  return delegate.read();
}
```

We also implemented basic versioning support (see requirement DR-U1 in deliverable report D1.1.1), in that for each extraction or transformation process we create a new named graph[17]. All processes depending on previously extracted or transformed data in the respective named graph use the newest such graph available by default. The version management data is also stored in a separate "graph management" graph in the Virtuoso store.

We will likely seek to improve the performance of this load step in collaboration with project partner OpenLink, e.g., by using optimised techniques that do not require the RDF serialisation needed here.

[15] Java Database Connectivity
[16] Java Virtual Machine
[17] see http://www.w3.org/TR/rdf-sparql-query/#namedGraphs

6 Vocabulary and Ontology Mappings

When building the extraction and transformation RDF mappings, we looked into vocabulary and ontology definitions that could be valuable to the scenario. Unfortunately, we did not find a single vocabulary or ontology which supports all the statements we have to represent. The W3C and SemanticWeb.org curated lists of trusted and highly used vocabularies[18][19] do not reference any tourism related ones, nor do Semantic Web search engines like Swoogle[20] and Sindice[21]. They only display very small RDF vocabularies, probably related to Semantic Web tutorials.

The closest candidates probably are a Travel Ontology[22] and another Travel Ontology[23], but they only incorporate basic classes. There are two thesauri, one of the World Tourism Organization (WTO)[24] and another of the UNESCO[25], but both are non-free and therefore will not be applied here.

Therefore, we used a set of different vocabularies and ontologies, the most important ones (except RDF, RDFS and OWL) listed below:

- Geo[26] for representing geocoordinates
- DC[27] for general information, particularly DC Terms for descriptions
- DBpedia[28] for many classes and well-known properties
- UMBEL[29] for other classes and hierarchy definitions

On top of that, we built another small ontology for integrating hotel types and additional tourism-specific information not well addressed in other vocabularies. A small part of this ontology, visualised by OntoViz, is shown in Figure 1.

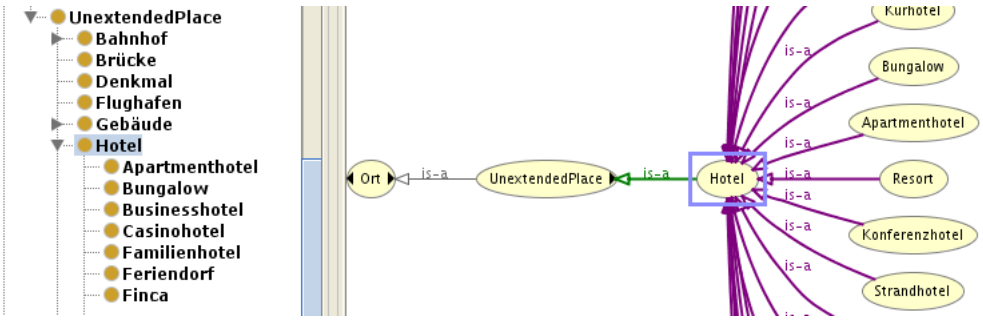

Figure 1: Snippet of additional ontology definitions

[18] http://www.w3.org/wiki/Lists_of_ontologies
[19] http://semanticweb.org/wiki/Ontology
[20] http://swoogle.umbc.edu/
[21] http://sindice.com
[22] http://www.schemaweb.info/schema/SchemaDetails.aspx?id=236, link not working as of 2013-10-25
[23] http://protege.cim3.net/file/pub/ontologies/travel/travel.owl
[24] http://pub.world-tourism.org:81/epages/Store.sf/?ObjectPath=/Shops/Infoshop/Products/1218/SubProducts/1218-1
[25] http://www2.ulcc.ac.uk/unesco/
[26] http://semanticweb.org/wiki/Basic_Geo_Vocabulary
[27] http://semanticweb.org/wiki/Dublin_Core
[28] http://dbpedia.org/Ontology
[29] http://umbel.org/specifications/vocabulary

7 Additional Data Sets

The data integration of private data sets and public data sources is one of the most challenging tasks. Unister has worked on such processes under the conditions of a real world scenario to be capable of evaluating the quality and quantity of the resulting data set.

In addition to the data sets provided in GeoKnow, we used the following data sets for testing the methods explained above:

- HotelsCombined[30] hotel and related information
- GIATA[31] data sets, for hotel and place information (including weather data etc.)
- DBpedia[32] RDF dumps (english, as well as international labels and abstracts)
- Geonames[33] RDF dump

The HotelsCombined information is provided as a regularly updated XML file, for which we implemented SpringBatch based parsing and processing. The GIATA information were stored in a relational database, which was then used as a data source in transformation processes, e.g., for enriching weather information of places.

We further used DBpedia and Geonames to provide detailed information on tourism-relevant places (points of interest), such as restaurants, museums, monuments, mountains etc. Currently, we only extract a subset of these very large data sets. Based on initial prototypes, we expect the interlinking of these data sets to be difficult at times. In some cases, resources should be regarded as a single entity in a tourism context (e.g., Berlin city and Berlin state), while other similar resources should be treated as distinct entities, such as the city of Zurich and the canton of Zurich.

Even though we only have preliminary results at this point, we can already see that this integration is a rather complex and error-prone task which requires much more work to improve the current results. The extent of potential errors, such as bad source data and erroneous reasoning, also calls for manual corrections and detailed logging of all processes. As a result, we currently have to deal with more than 40GB of log files (no TRACE logging) for running all prototype data integration jobs. We will collaborate in GeoKnow work package WP4 (particularly in Task 4.3) on such integration and data evolution issues.

[30] http://www.hotelscombined.de/Affiliates.aspx?languageCode=EN
[31] http://www.giata.de/en.html
[32] http://dbpedia.org
[33] http://www.geonames.org/ontology/documentation.html

8 Summary

The potential methods for extracting, transforming and loading source data in the "GeoKnow for E-Commerce" scenario and our experiences when applying them have been described in this report. We focused on several methods for providing relational sources on the fly as RDF (D2RQ, Sparqlify) and frameworks which can be used for generating RDF in ETL processes (Spring Batch, Scriptella).

We further discussed how the data has to be transformed, our infrastructure for loading the resulting RDF. Additional vocabularies and data sets which were used in this process have also been described, also commenting on those that could not be applied for various reasons. The final section on additional data sets refers to the upcoming tasks of integrating private and public data sets and respective issues.