QROWD - Because Big Data Integration is Humanly Possible

Innovation action

D4.3 – Linked Data Generation

<table>
<thead>
<tr>
<th>Author/s</th>
<th>Claus Stadler (InfAI)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gordian Dziwis (InfAI)</td>
</tr>
<tr>
<td></td>
<td>Patrick Westphal (InfAI)</td>
</tr>
<tr>
<td>Due date</td>
<td>30.11.2018</td>
</tr>
<tr>
<td>Actual delivery date</td>
<td>14.12.2018</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
<tr>
<td>Dissemination level</td>
<td>PU</td>
</tr>
<tr>
<td>Status</td>
<td>FINAL</td>
</tr>
</tbody>
</table>
## History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason</th>
<th>Revised By</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>2018-09-03</td>
<td>Initial structure and content</td>
<td>Claus Stadler</td>
</tr>
<tr>
<td>0.2</td>
<td>2018-09-18</td>
<td>Description of Sparql-Integrate and Dcat-Suite</td>
<td>Claus Stadler</td>
</tr>
<tr>
<td>0.3</td>
<td>2018-09-28</td>
<td>Description of NiFi Integration</td>
<td>Gordian Dziwis</td>
</tr>
<tr>
<td>0.4</td>
<td>2018-10-04</td>
<td>Description of TomTom dataset</td>
<td>Patrick Westphal</td>
</tr>
<tr>
<td>0.5</td>
<td>2018-10-16</td>
<td>Description of LinkedGeoData</td>
<td>Claus Stadler</td>
</tr>
<tr>
<td></td>
<td>2018-11-08</td>
<td>Internal review of draft</td>
<td>Enrico Bignotti</td>
</tr>
<tr>
<td>0.6</td>
<td>2018-11-16</td>
<td>Revision according to review</td>
<td>Claus Stadler</td>
</tr>
<tr>
<td>0.7</td>
<td>2018-11-20</td>
<td>Updated statistics</td>
<td>Claus Stadler</td>
</tr>
<tr>
<td></td>
<td>2018-11-22</td>
<td>Final internal review</td>
<td>Enrico Bignotti</td>
</tr>
<tr>
<td>0.8</td>
<td>2018-11-26</td>
<td>Minor revisions</td>
<td>Claus Stadler</td>
</tr>
<tr>
<td></td>
<td>2018-11-30</td>
<td>Several requests for clarifications</td>
<td>Luis-Daniel Ibáñez</td>
</tr>
<tr>
<td>1.0</td>
<td>2018-12-12</td>
<td>Added clarifications</td>
<td>Claus Stadler</td>
</tr>
<tr>
<td></td>
<td>2018-12-14</td>
<td>Approval</td>
<td>Luis-Daniel Ibáñez</td>
</tr>
</tbody>
</table>

Project co-funded under Horizon 2020 Research and Innovation EU programme, grant agreement no. 732194
Table of Contents

List of Figures 5
List of Listings 5
List of Abbreviations 5
Abstract 6
Executive Summary 7
1. Introduction 8
2. Preliminaries 10
3 Large-Scale RDF Data Generation 11
   3.1 RDFization of TomTom MultiNet and MultiNet POI 11
      3.1.1 Data Conversion 12
      3.1.2 Ontology Development 12
   3.2 Linked Data Generation From OpenStreetMap 13
4 Linked Data Generation with Sparql-Integrate 17
   4.1 Concepts 18
      4.1.1 Specification of Workloads and Result Graphs 20
      4.1.1 Using Files as input for Workload specifications 21
   4.2 Command-Line Interface Tool 22
   4.3 Linked Data Generation Use Cases 22
      4.3.1 Bike Racks 23
      4.3.2 Timetables for mode detection 25
5 Dataset Retrieval and Publishing with DCAT Suite 27
   5.1 The DCAT Suite Command Line Tool 28
   5.2 Bridging CKAN and DCAT-AP 28
      5.2.2 DCAT Suite Command Line Client 29
6. Apache Nifi Integration 30
6 Related Work 31
   6.1 Linked Data Generation 31
   6.2 Data Catalogue Models 31
7 Conclusions and Future Work 32
References 32
**LIST OF FIGURES**

Figure 1: Linked Data Generation within QROWD  
Figure 2: Ontology metrics provided by the Protégé ontology editor  
Figure 3: Proposed workflow design with Sparql-Integrate  
Figure 4: Overview of the DCAT Suite Components  
Figure 5: A NiFi workflow for uploading a set of distributions

**LIST OF LISTINGS**

Listing 1: Newly introduced DCAT metadata for LinkedGeoData  
Listing 2: SPARQL query that fetches a remote JSON object and extracts an attribute  
Listing 3: Unnesting a JSON-array with a property function  
Listing 4: Unnesting XML based on an XPath expression  
Listing 5: Processing CSV rows into JSON objects  
Listing 6: Federating queries to all files in a folder and its sub folders  
Listing 7: Converting coordinates collected using crowdsourcing with VCE to RDF  
Listing 8: Excerpt of a KML file about bike racks in Trento  
Listing 9: Declaration of mappings and referencing them in queries  
Listing 10: Zipping arrays by index with Sparql-Integrate  
Listing 11: CKAN-DCAT mapping specifications in RDF

**LIST OF ABBREVIATIONS**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>crowdsourcing</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>DCAT</td>
<td>Data Catalog Vocabulary</td>
</tr>
<tr>
<td>DCAT-AP</td>
<td>DCAT application profile for data portals in Europe</td>
</tr>
<tr>
<td>SPARQL</td>
<td>SPARQL protocol and RDF query language</td>
</tr>
<tr>
<td>OSM</td>
<td>OpenStreetMap</td>
</tr>
<tr>
<td>LGD</td>
<td>LinkedGeoData</td>
</tr>
<tr>
<td>TTL</td>
<td>Turtle - an RDF syntax</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript object notation - a data format</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible markup Language - a data format</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma-separated values - a data format</td>
</tr>
</tbody>
</table>
ABSTRACT
In this deliverable, we present our approach to generate Linked Data from heterogeneous data sources. These efforts include specialized transformations of large-scale data sources, namely TomTom MultiNet and OpenStreetMap, and an innovative approach for converting almost any data format to RDF. While several mapping languages have been devised for the purpose of Linked Data generation, we discovered that SPARQL itself is already capable of solving this task effectively and efficiently - provided that certain function extensions were added. Whereas SPARQL natively supports transforming RDF data (to RDF), in this work, we introduce several extensions for processing other formats, namely XML, JSON and CSV data. Furthermore, SPARQL features for interacting with the file system are introduced, enabling idiomatic access of RDFization processes to information stored in files. The contributions of this work are: (a) the transformation of two large-scale data sources, (b) plugins that extend Apache Jena’s SPARQL system with additional functionality, and (c) command-line tools and Apache NiFi plugins for carrying out data conversion and data publishing to a CKAN dataset catalog system.
EXECUTIVE SUMMARY

This document is aimed at developers, researchers and audiences with an interest in topics related to data integration, data publishing, and the intersection of Linked Data and data catalogues. Due to the nature of QROWD, we particularly emphasize that this includes Smart Cities and public administration units striving to become such, as this deliverable provides input on how non-semantic data assets can be converted to machine-readable Linked Data in order to ease subsequent use.

The outcomes of this deliverable are:
- Linked Data generation from two large scale data sources, namely TomTom and OpenStreetMap (OSM) for regions in Italy, especially Trento,
- two complementary software systems for Linked Data generation and (Linked) Data publishing using DCAT(-AP), as well as
- the descriptions of QROWD use cases where the data either is already put to use or its use is subject to evaluation.

The RDF conversions for TomTom/Italy and OpenStreetMap/North Italy resulted in 750 million and OSM 1.5 billion triples, respectively. Several 10K triples were further obtained from conversion of about 30 datasets from the Trento dataset catalog.

The dataset catalogue assembled in WP4 (D4.1) forms the basis for the generation of Linked Data. Output datasets have been successfully applied in order to create reference data related to the use case UC3 ‘Completing Mobility Infrastructure’ on the example of bike racks, whereas other data, such as bus routes and timetables, are most likely to contribute to improving the modal split algorithms of WP6; possibly via further integration by means of interlinking and fusion via WP5.
1. **INTRODUCTION**

In this deliverable, we report on our Linked Data generation efforts. This comprises RDF conversion of datasets and the development of novel tools. Linked Data is highly advantageous to data integration tasks, such as linking and fusion (WP5), due to the fact that all entities of relevance have referenceable globally unique IDs, which allows adding and removing individual semantic relations between them as needed. Even in cases where datasets are published in non-RDF formats (such as to retain backward compatibility), Linked Data is a powerful intermediate representation capable of capturing both data and metadata in a uniform model.

For example, in regard to use case UC1 ‘Modal Split’ (D2.2), this technology is aimed to improve the analytics of WP6 as input to the developed symbolic machine learning approaches. Concretely, Linked Data generation is actively applied for use case UC3 ‘Completing Information About Mobility Infrastructure’ as a basis to combine information from different sources.

OpenStreetMap (OSM), TomTom MultiNet and MultiNet POI are project-relevant large scale data sources for road network and Point of Interest (POI) information. Furthermore, at the time of writing, the live document of relevant datasets with Trento-related ownership (initially published in D4.1, further updates tracked in a spreadsheet) has grown to 80 candidates. Out of those, 23 are in-use, and several more datasets with potential future use have been identified. For example, timetable information of public transport information may prove useful for the mode detection / modal split system (WP6) for validation purposes or improving confidence scores.

In order to facilitate the integration of data from these sources, we transformed several of these datasets to RDF. With the exception of TomTom MultiNet and MultiNet POI, all datasets and tooling are publicly available. The following summarizes datasets converted to RDF together with the approach used to produce them:

- For OSM, we could reuse the RDF conversion tool of our LinkedGeoData community project in order to extract an excerpt for the Trento region. Some adjustment of mapping definitions was required to improve the data quality.
- For TomTom MultiNet and MultiNet POI, Python scripts were created that yield RDF output in accordance with the accompanying data and format specifications. The input and output datasets are confidential.
- The Trento-related datasets come in heterogeneous formats. Drawing from the experience with RDF conversion of the first two datasets, we sought to devise a uniform solution to this problem, which lead to the development of Sparql-Integrate.

With respect to tooling, the following developments are outcomes of our spent efforts:

- **SPARQL-Integrate** is our system for converting CSV, XML and JSON data to RDF. The chosen approach is simple, effective and extensible: We introduce new functions to the SPARQL engine of the popular Apache Jena Semantic
Linked Data Generation

Web Toolkit\(^1\) using its plugin system. Consequently, our plugin can be easily integrated into existing Jena-based software in order to make them capable of processing heterogeneous data.

- **DCAT Suite** is our system for publishing and downloading data from data catalogues. In the case of RDF data, it supports interaction with triple stores.

We provide appropriate packaging to enable their use as Java libraries, command line tools and NiFi processors. With the exception of confidential data by TomTom, all resources related to this deliverable are publicly available:

- Sparql-Integrate: [https://github.com/QROWD/Sparql-Integrate](https://github.com/QROWD/Sparql-Integrate)
- jena-sparql-api-sparql-ext: Our library where we implemented the SPARQL extensions and which powers Sparql-Integrate:
  [https://github.com/SmartDataAnalytics/jena-sparql-api/tree/master/jena-sparql-api-sparql-ext](https://github.com/SmartDataAnalytics/jena-sparql-api/tree/master/jena-sparql-api-sparql-ext)
- Dcat-Suite: [https://github.com/QROWD/dcat-suite](https://github.com/QROWD/dcat-suite)
- RDFization repo: [https://github.com/QROWD/QROWD-QROWD-RDF-Data-Integration](https://github.com/QROWD/QROWD-QROWD-RDF-Data-Integration)
- LinkedGeoData: [https://github.com/GeoKnow/LinkedGeoData](https://github.com/GeoKnow/LinkedGeoData)

Figure 1 shows the position of Linked Data generation within QROWD, especially in relation to acquisition of source data, RDFization and and its input to data integration.

\(^1\) [https://jena.apache.org/](https://jena.apache.org/)
The remainder of this document is structured as follows: Section 2 briefly introduces relevant key technologies. RDF generation from large-scale data sources is presented in Section 3. Afterwards, we present the Sparql-Integrate toolkit for Linked Data generation in Section 4. Subsequently, Section 5 presents the Dcat-Suite system for Linked Data publishing. A brief summary of our NiFi integration efforts are given in Section 6. Finally, related work is discussed in Section 7 and we conclude in Section 8 where we also point out future work.

2. PRELIMINARIES
Linked Data Generation

In this section we give a brief overview of key technologies relevant to this deliverable.

**Apache Jena**
Apache Jena\(^2\) is a popular Semantic Web toolkit which is under active development for more than a decade. It features a plethora of libraries and tools, such as an RDF API, parsers and writers for a vast amount of RDF formats, an extensible SPARQL engine with configurable levels of SPARQL standard compliance, in-memory and disk-based triple stores, and Web services.

**CKAN**
The *Comprehensive Knowledge Archive Network*\(^3\) (CKAN) is the Open Source data catalog software that powers many leading Open Data data portals, such as the U.S. Government’s Open Data Portal (~300K datasets)\(^4\) and the European Data Portal (~866K datasets)\(^5\). A CKAN instance for QROWD has been deployed for managing datasets produced in the project.

**DCAT and DCAT-AP**
The *Data Catalog Vocabulary (DCAT)* is a W3C standard vocabulary\(^6\) with the aim to facilitate interoperability between data catalogs published on the Web. The most notable conceptual contribution is the distinction between datasets and distributions: According to the standard, a distribution “represents a specific available form of a dataset”, such as a CSV file or an API, whereas a dataset is a “collection of data, published or curated by a single agent, and available for access or download in one or more formats”. Hence, a dataset comprises one or more distributions.

**Apache NiFi**
Apache NiFi\(^7\) is the workflow engine agreed upon by the QROWD consortium in order to realize many data flows within the project. NiFi features support for static and streaming data, a visual workflow builder, a large and extensible library of plugins, and advanced monitoring facilities, such as for throughput.

### 3 LARGE-SCALE RDF DATA GENERATION

In this section, we describe how we created RDF data from TomTom MultiNet and OpenStreetMap for the use case regions.

#### 3.1 RDFization of TomTom MultiNet and MultiNet POI
TomTom MultiNet and MultiNet POI are products of TomTom N.V. which contain map data covering the whole globe. The data provides geographic, topographic, and demographic data with a focus on transportation networks. The data is provided in

---

\(^2\) [https://jena.apache.org/](https://jena.apache.org/)
\(^3\) [https://ckan.org/](https://ckan.org/)
\(^4\) [https://catalog.data.gov/dataset](https://catalog.data.gov/dataset)
\(^6\) [https://www.w3.org/TR/vocab-dcat/](https://www.w3.org/TR/vocab-dcat/)
\(^7\) [https://nifi.apache.org/](https://nifi.apache.org/)
different formats like GDF-AS, GDF-AR\textsuperscript{8}, Shapefile\textsuperscript{9}, and Oracle Shapefile Loader (OSL). The QROWD consortium got access to the TomTom MultiNet and MultiNet POI data of whole Italy which is partitioned into 20 sub-datasets of same structure. However, the respective license agreement forbids to make any of the MultiNet and MultiNet POI data, as well as any derived data publicly available.

### 3.1.1 Data Conversion

For the data conversion we focused on the OSL format which stores the information in a relational structure which seemed easiest for parsing and data conversion. For each aspect that is stored in MultiNet and MultiNet POI a dedicated schema is used. This sums up to 1613 different schemas (i.e. ‘file types’) for MultiNet and 26 for MultiNet POI. Most of the schemas are explained on 1648 pages of textual documentation, spread across 8 PDF files shipped with these TomTom data products. These data and format specifications also cover the meaning of data values in respective contexts, as throughout all these data files mostly numbers were used to encode meaning. Besides numeric identifiers, categorical data represented by integer numbers, also bit fields were used. To give a (made up) example the 23th column in OSL files named mn\_abc.dat (one for each data partition of whole Italy) could encode certain road conditions on road segments, such that the binary representation of the stored number encodes that

- the road segment has a cycle lane if the first bit (bit position zero) is set
- you are not allowed to overtake on this road segment if the second bit (bit position one) is set
- there is no sidewalk on this road segment if the third bit (bit position two) is set etc.

A road segment with a cycle lane (bit position 0 equals 1), without any restrictions on overtaking (bit position 1 equals 0), and without a sidewalk (bit position 2 equals 1) would then be encoded by the integer number 5 which is $1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2$. Bit fields in other columns and/or schemas are likely to have a completely different meaning. Identifiers are not unique throughout the whole dataset but depend on the respective type of the entity identified. Thus, identifiers may be reused e.g. for roads, junctions, tunnels etc.

Given these circumstances a general or even automatic conversion approach was not feasible such that this conversion was done manually with a schema-by-schema approach according to the documentation provided by TomTom. The OSL file input, per-schema data conversion and RDF output is established in a Python software project (which cannot be made public as well since it exposes all the schema information which is property of TomTom). The RDF data generated (for whole Italy) sums up to about 750 million triples, which amounts to 37 GB of uncompressed Turtle files. This data covers the basic information we considered useful for the task T6.3 ‘Spatio-temporal analytics for cities’ where we aim to learn symbolic and meaningful classifiers for different transportation modes used on commutes. These classifiers describe the distinctive patterns of each considered mode by means of relations to spatial features like those contained in MultiNet and MultiNet POI. Since not all information contained in MultiNet and MultiNet POI seems relevant for this task many of the 1613 schemas were omitted, but might be introduced later, as needed.


3.1.2 Ontology Development

Along with the data conversion we developed an OWL ontology describing the TomTom MultiNet and MultiNet POI data. This ontology on the one hand side serves as a conceptual description shared across all schemas. Besides this it shall provide a rich ontological background for learning OWL class expressions by means of the DL-Learner\(^\text{10}\) framework. These learned class expressions shall serve as symbolic classifiers, i.e. human-understandable, logical descriptions of what are characteristic properties of commutes performed using a certain means of transportation.

The developed ontology is anchored in the Basic Formal Ontology (BFO)\(^\text{11}\) following best practices in ontology development, and thoroughly provided with human readable labels and descriptions following the Aristotelian definition structure. Since this ontology, again, mirrors internals of the TomTom MultiNet and MultiNet POI data product, we are not allowed to make it publicly available. The properties of the developed ontology in its current state is shown in Figure 2.

![Figure 2: Ontology metrics provided by the Protégé ontology editor](http://basic-formal-ontology.org/)

Even though this ontology is complete in the sense that it provides descriptions for all the extracted data described in the previous sub-section, we still expect future

\(^{10}\) [http://dl-learner.org/](http://dl-learner.org/)

\(^{11}\) [http://basic-formal-ontology.org/](http://basic-formal-ontology.org/)
Linked Data Generation

revisions. These can be caused by inconsistencies discovered while actually working with the ontology (e.g. in T6.3) that were not visible during the ontology development and data conversion phase. To give an example, we first treated junctions as point features and only later discovered certain sub-types of junctions that were modeled as line features in the TomTom MultiNet data. Another reason for future changes might be performed to reduce the reasoning complexity.

3.2 Linked Data Generation From OpenStreetMap

LinkedGeoData\(^1\) (LGD) is a community effort to add a spatial dimension to the Web of Data / Semantic Web. LGD uses the information collected by the OpenStreetMap project and makes it available as an RDF knowledge base according to the Linked Data principles. In the GeoKnow project\(^2\) (2013-2015), we created fundamental tooling\(^3\) to accomplish the RDFization of OSM data, which InfAI maintained and could thus readily reuse. Contributions from QROWN are enhancement of the dataset creation process to also yield DCAT metadata and improvement of the quality of OSM-to-RDF mappings for use-case relevant domains, such as the geographic description of bike racks and disabled parking. The core component of LGD is an Ontology-Based Data Access (OBDA) system, which rewrites SPARQL queries via manually crafted mapping information to SQL queries over an underlying physical OSM PostgreSQL\(^4\) database.

An external service that processes OSM data into partitions and makes them available for download is GeoFabrik\(^5\). The data it publishes about Trento is contained in the partition italy/nord-est\(^6\), however it also lies quite close to the region boundary italy/nord-ovest\(^7\). Hence, for RDF extraction we used both datasets.

Fortunately, OSM provides the Osmosis tool\(^8\), which makes the process of merging multiple region files into a single one very easy:

```bash
osmosis \
--read-pbf nord-est-latest.osm.pbf --read-pbf nord-ovest-latest.osm.pbf \
--merge --write-pbf > nord-latest.pbf
```

Complementary, our LGD tool makes it easy to set up an RDFization-ready OSM database from a single dataset:

```bash
lgd-createdb -h localhost -d lgd_italy_nord -U postgres -W pwd -f 
nord-latest.pbf 
# Note: Instead of 'latest' we recommend to use an appropriate timestamp
```

---

\(^1\) [http://linkedgeodata.org/](http://linkedgeodata.org/)
\(^2\) [http://geoknow.eu](http://geoknow.eu)
\(^3\) [https://github.com/GeoKnow/LinkedGeoData](https://github.com/GeoKnow/LinkedGeoData)
\(^4\) [https://www.postgresql.org/](https://www.postgresql.org/)
\(^5\) [https://www.geofabrik.de/](https://www.geofabrik.de/)
\(^6\) [http://download.geofabrik.de/europe/italy/nord-est.html](http://download.geofabrik.de/europe/italy/nord-est.html)
\(^7\) [http://download.geofabrik.de/europe/italy/nord-ovest.html](http://download.geofabrik.de/europe/italy/nord-ovest.html)
\(^8\) [https://wiki.openstreetmap.org/wiki/Osmosis](https://wiki.openstreetmap.org/wiki/Osmosis)
Linked Data Generation

We can now use our `sparqlify-tool` to run SPARQL queries over the set up database via a given set of mappings:

```
sparqlify-tool -d lgd_italy_nord -h localhost -U postgres -W pwd \
-m /usr/share/linkedgeodata/sml/LinkedGeoData-Triplify-IndividualViews.sml \
-Q 'CONSTRUCT WHERE { ?s ?p ?o }'
```

The example above queries for all triples and thus creates a full dump of the dataset. However, the data for this comparatively small region comprises about 100 million point entities. As in the RDF each point carries at least 3 types and a geometry, this already quadruples the number of triples. In total, 1.5 billion triples are generated only for north Italy. As such a large number is difficult to handle, LGD also comes with a dump creation tool that is capable of creating individual dataset files for specific OSM POI types, such as bike racks\(^\text{20}\). In this case, it yields only ~20K triples.

Listing 1 shows an example output of the newly added DCAT metadata generation.

```
@prefix dct: <http://purl.org/dc/terms/> .
@prefix dcat: <http://www.w3.org/ns/dcat#>.

    dct:title "LinkedGeoData italy-north 2018-04-04" ;
    dcatx:defaultGraphGroup
    dcat:distribution

    dct:title "osm-bremen-2018-04-04-nodes-abutters" ;
    dct:description "RDF dataset containing OSM nodes of type bicycle-parking" ;
    dct:downloadURL <osm-bremen-2018-04-04-nodes-bicycle-parking.sorted.nt.bz2> ;
    dct:format "RDF" ;
    dcatx:defaultGraph
.
```

**Listing 1: Newly introduced DCAT metadata for LinkedGeoData**

Note that `dcatx:defaultGraphGroup` and `dcatx:defaultGraph` are our extensions to specify into which named graphs the distributions should be loaded by default. In general, the recommended interpretation is, that RDF datasets loaded into a triple store based on such a specification should be accessible under the graph given in the graph group. Some triple stores support 'virtual' graph names that get eventually expanded (possibly recursively) to physical graph names. In such a case, the individual distributions are best to be loaded into their own physical graphs denoted by default graph.

The obtained LGD datasets were deployed to our QROWD CKAN instance at [http://ckan.qrowd.aksw.org/dataset?q=LinkedGeoData](http://ckan.qrowd.aksw.org/dataset?q=LinkedGeoData) using the DCAT Suite tool (see Section 5).
4 LINKED DATA GENERATION WITH SPARQL-INTEGRATE

Sparql-Integrate is a set of components for processing heterogeneous data with SPARQL. The development of the tool was motivated by the challenge of how to facilitate the RDFization of as many datasets in the QROWD dataset catalog (D4.1 ‘Data Catalog’) as possible in a flexible, uniform, and ultimately maintainable and sustainable way.

Sparql-Integrate is backed by our SPARQL extensions for Apache Jena plugin project. In this deliverable we present the highlights of our new additions in the context of QROWD. For the complete function reference please refer to: https://github.com/SmartDataAnalytics/jena-sparql-api/tree/master/jena-sparql-api-sparql-ext

Sparql-Integrate’s core is formed by a set of SPARQL function and datatype extensions, of which some are demonstrated in the query below:

```sparql
SELECT ?title {
    # Retrieve a text/plain representation of the given URL and bind it to ?o
    <https://json-ld.org/schemas/jsonld-schema.json> url:text ?o
    
    # Process the string representation into a literal of json datatype
    BIND(STRDT(?o, xsd:json) AS ?json)
    
    # Extract the title attribute of the json object and bind it to ?title
    BIND(json:path(?json, "$.title") AS ?title)
}
```

<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Schema for JSON-LD&quot;</td>
</tr>
</tbody>
</table>

Listing 2: SPARQL query that fetches a remote JSON object and extracts an attribute

Note, that the Sparql-Integrate tooling has several common prefixes preconfigured - including those of the RDFa initial context\(^{21}\). Hence, these prefixes can be used in SPARQL statements without explicit declaration, and the system makes use of them in applicable output formats.

For simplicity - especially in regard to reducing the cognitive load - we added our json and xml datatype implementations next to all standard datatype definitions into

\(^{21}\) [https://www.w3.org/2011/rdfa-context/rdfa-1.1](https://www.w3.org/2011/rdfa-context/rdfa-1.1)
Linked Data Generation

the xsd namespace. We want to note that, this is subject to change as adding
definitions to namespaces not under one’s own authority is usually frowned upon.

In the remainder of this section, we first discuss Sparql-Integrate’s concepts,
especially with respect to workflows. Subsequently, we demonstrate some of its
features using several examples, both real and contrived (for brevity).

4.1 Concepts

The introduction of SPARQL functions with the purpose to extract values from
non-RDF objects - such as JSON attributes - seems natural. The main problem that
needs to be tackled is how to perform iteration over elements, such as rows of a
CSV file, JSON arrays, or elements of an XML document. Fortunately, the Apache
Jena framework supports two type of functions:

- Conventional functions are objects that take zero or more RDF terms as
arguments and compute a single resulting RDF term. Typically, they are used
to construct selection and projection expressions. For example, STR(?arg) is
a function that yields a string representation of its argument.
- Property functions are syntactically equivalent to conventional triple patterns
with a specific predicate. However, instead of denoting a set of bindings
based on the matching triples in a dataset, they are actually functions that can
compute sets of bindings by other means. Consequently, IRIs of property
functions typically are not actually present in datasets.

Property functions can thus be used for e.g. unnesting collections into a set of result
rows with one item per row, as shown in Listing 3.

```
SELECT ?item ?index {
  "[1, 2, 3, 4]"^^xsd:json json:unnest (?item ?index)
}
```

<table>
<thead>
<tr>
<th>item</th>
<th>index</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;a&quot;</td>
<td>0</td>
</tr>
<tr>
<td>&quot;b&quot;</td>
<td>1</td>
</tr>
<tr>
<td>&quot;c&quot;</td>
<td>2</td>
</tr>
</tbody>
</table>

Listing 3: Unnesting a JSON-array with a property function

For XML and CSV the approaches are quite analogous. For XML the main difference
is that unnesting is performed using an XPath expression that yields a sequence of
matching XML nodes, as shown in Listing 4. Accessing attributes as strings is
Linked Data Generation

accomplished via a SPARQL function that takes an XML literal and an XPath expression as arguments. Conversion of strings to e.g. numeric types can be done using standard SPARQL features.

```sparql
SELECT ?str ?int {
  ""
  <ul>
    <li id="1" />
    <li id="2" />
  </ul>"^^xsd:xml
  xml:unnest ("//*[local-name()='li']" ?o)
  BIND(xml:path(?o, "//*[id]") AS ?str)
  BIND(xsd:integer(?str) AS ?int)
}
```

<table>
<thead>
<tr>
<th>str</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;1&quot;</td>
<td>1</td>
</tr>
<tr>
<td>&quot;2&quot;</td>
<td>2</td>
</tr>
</tbody>
</table>

---

**Listing 4: Unnesting XML based on an XPath expression**

For CSV data, we provide a CSV parse property function: Each row of the CSV data referred to by the argument given in its subject position is processed into a JSON object. If the subject is an IRI, an attempt to open a byte stream to the resource is made. Otherwise, data can be inlined using a string literal. An option string can be provided to control the parsing of CSV data. For example ‘-h’ indicates to use the first row as headers, which yields JSON objects with keys according to the CSV headings as shown in Listing 5.

```sparql
SELECT * {
  """fn,ln
  Mary,Major
  John,Doe""" csv:parse (?rowJson "excel -h")
}
```

# CSV data is streamed if read from a URL:
SELECT * { <data.csv> csv:parse (?rowJson "excel -h") }

---
The SPARQL CONSTRUCT query form is a succinct and powerful way to express which triples to generate from an underlying result set. However, conventions and design patterns are needed to use SPARQL queries to create workflows.

### 4.1.1 Specification of Workloads and Result Graphs
A Sparql-Integrate process is at the core a sequence of SPARQL queries, which we refer to as “sparql script”. However, often the same Linked Data generation process can be applied to an arbitrary number of input datasets. For example, time table information published with the same schema can be transformed using a single process specification as long as it caters to different input file names and metadata values, such as publishing date.

Hence we can distinguish between these phases:

1. **Workload specification**: The goal of this phase is to obtain an RDF graph about what to RDFize, which sets it apart from the subsequent processing phase, which is concerned about how to perform the RDFization. This decoupling allows the same RDFification process to be used in conjunction with different approaches to workload specification, as the latter are highly dependent on the environment: On the command line, typically a script is used to download source files, collect metadata and on that basis set up what to RDFize. An idiomatic approach how Sparql-Integrate can be used for this purpose is presented in Section 4.1.2. In the Apache NiFi context, this is achieved by appropriate configuration of NiFi processors. Conceptually, in both cases, we read in an RDF file containing DCAT information (e.g. named dcat.ttl) and create new ‘workload’ resources from all distributions matching certain criteria. As there can be multiple workloads - and thus multiple result outputs - it is useful to annotate each workload with the result graph that will eventually hold the corresponding generated triples.

2. **Process specification**: A sparql script that processes all workloads and generates output triples. For example, generation of Linked Data from JSON objects would happen in this phase.

3. **Output specification**: While processes can directly produce output triples and quads (using CONSTRUCT queries), this phase adds the opportunity to yield finalizing output from the state of the backing triple store at the end of the process.

Note, that the result of a Sparql-Integrate process is merely a set of quads derived from a set of sources. Yet, the usual way for distribution of data is as one or more
Linked Data Generation

separate files for each source. Hence, it is necessary to “disentangle” which of the output quads corresponds to a file for distribution. For this purpose we define the following convention: First and foremost, processes should be designed to generate DCAT metadata in the default graph. On that basis, every URL of a graph that matches the URL given as a value for a $dcat:accessURL$ is eligible for being exported as a file of a DCAT distribution. The Dcat-Suite provides the ‘expand’ command for this purpose, which will create files for these graphs’ content and output new DCAT metadata where the original $dcat:accessURL$s are replaced with $dcat:downloadURL$s with relative paths to the expanded files.

In certain cases, data and metadata preparation can be automated using domain specific scripts, such as downloading distributions via a REST API. The “create-workloads.script” box in Figure 3 represents these cases.

![Figure 3: Proposed workflow design with Sparql-Integrate](image)

4.1.2 Using Files as input for Workload specifications

A typical static data conversion project processes one or more source data files from the disk. Our SPARQL-idiomatic solutions are demonstrated in Listing 6 and the main building blocks are summarized as follows:

- A property function `<basePath> fs:find ?outputPath` that recursively lists all sub-folders and files for the given base path in its subject position
- A function `xsd:boolean fs:probeRdf(URI)` that probes whether the given argument is an RDF file by attempting to read a triple using various parsers.
- A custom SERVICE processor that - before attempting to send queries off to remote endpoints - first checks whether the referenced service is actually an RDF file. If so, the file will be loaded into an (in-memory) store and the pattern of the SERVICE clause will evaluated against it.

This mechanism can on the one hand be used to create workloads from e.g. all JSON files in a directory, as well as reading in metadata from accompanying RDF files, such as DCAT descriptions.

For example, consider two files (one nested in a sub-folder named `folder`) each containing a simple triple. The query below will recursively find all files in the current directory. A filter is applied to only retain those that contain RDF content. The modified SERVICE processor will then run the provided pattern on every file.
Input files
# ./a.nt
<s> <p> <o> .
# ./folder/b.nt
<x> <y> <z> .

Query over the file system
# ./list-rdf.sparql
SELECT * {
<> fs:find ?file
FILTER(fs:probeRdf(?file))
SERVICE ?file {
?s ?p ?o
}
}

Result

<table>
<thead>
<tr>
<th>file</th>
<th>s</th>
<th>p</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="">file:///.../a.nt</a></td>
<td>&lt;s&gt;</td>
<td>&lt;p&gt;</td>
<td>&lt;o&gt;</td>
</tr>
<tr>
<td><a href="">file:///.../folder/b.nt</a></td>
<td>&lt;x&gt;</td>
<td>&lt;y&gt;</td>
<td>&lt;z&gt;</td>
</tr>
</tbody>
</table>

Listing 6: Federating queries to all files in a folder and its sub folders

4.2 Command-Line Interface Tool

The command line interface (CLI) tool of Sparql-Integrate makes it very easy to run a sequence of SPARQL queries. The following conventions apply:

- CONSTRUCT queries produce machine readable dataset output
- SELECT queries produce textual output (such as for debugging)
- INSERT / UPDATE / LOAD statements modify the in-memory store that backs the SPARQL query execution.

The introductory example in Listing 2 can be run with using the following command if saved to a file:

csparql-integrate example.sparql [... fileN.sparql]

Specification of multiple files is conceptually treated as a concatenation of their content with one important quirk: The base URL of every query is adjusted to the
Linked Data Generation

corresponding directory containing the .sparql file. This allows for relative references to other files such as the datasets which to transform.

All output formats provided by the Apache Jena framework are supported. The snippet below shows the result of changing the initial query to a CONSTRUCT query, and requesting output to be generated in the ‘turtle/pretty’ format:

```sparql
example.sparql: CONSTRUCT { eg:jsonld-schema rdfs:label ?title }
WHERE { # Body of Listing 2 }
cli invocation: sparql-integrate -w turtle/pretty example.sparql
```

For line based output formats - i.e. n-triples and n-quads - the output is streamed which may exhibit significantly lower memory consumption than other formats. For CSV, we provide a streaming reader that enables Linked Data generation for arbitrary large input.

4.3 Linked Data Generation Use Cases

In this section, we show selected use cases of datasets converted to RDF using Sparql-Integrate. In general, Linked Data generation is an ongoing effort. At the time of writing, about 30 Linked Data datasets - including appropriate DCAT metadata - have been created using this approach. Our data integration efforts are published at https://github.com/QROWD/QROWD-RDF-Data-Integration

4.3.1 Bike Racks

A use case demonstrating the effectiveness of crowdsourcing approaches and technology developed in QROWD was devised on the example of bike racks in Trento: Crowdsourcing experiments were conducted with the Virtual City Explorer (VCE)\(^{22}\) in order to collect information about bike racks, most notably location, type, capacity and depiction. In addition, a dataset on this topic was extracted from OpenStreetMap and the Municipality Trento assembled their own official dataset, resulting in data from three sources subject to consolidation using interlinking and fusion in WP5. As a prerequisite to this effort, all involved data was converted to RDF.

In the case of VCE, the input data is essentially an array of JSON objects with locations gathered from crowd workers, similar to example shown in Listing 7.

```
[ { id: 1, x: 46.06978628330533, y: 11.120105733165468 }, ... ]
```

The conversion is accomplished using the following query:

```
CONSTRUCT {
```

\(^{22}\) Cf. deliverable D3.2 ‘Crowdsourcing Services’ for more information
Linked Data Generation

```sql
?s a lgdo:BicycleParking ;
    geo:long ?x ; geo:lat ?y
}

{vce_trento_bikeracks.json} url:text ?src
BIND(STRDT(?src, xsd:json) As ?json)
?json json:unnest (?item ?index)
BIND(json:path(?item, "$.x") AS ?x)
BIND(json:path(?item, "$.y") AS ?y)

# IRI generation
BIND("http://qrowd-project.eu/resource/vce/trento/bikerack/", ?ns)
BIND(CONCAT(STR(?ns, json:path(?item, "$.id"))) AS ?s)
}
```

**Listing 7: Converting coordinates collected using crowdsourcing with VCE to RDF**

Note, that crowdsourcing experiments are being conducted with the VCE for collecting additional information, such as the maximum and available capacity of bike racks. Exposing this information as RDF from further JSON fields works analogously.

The official data from the Municipality of Trento with respect to bicycle parking is published in several formats, namely GML, KML and SHP. We chose the KML version which contains entries similar to that shown in Listing 8:

```xml
<Placemark>
    <name>passaggio Peterlongo n.posti 10 - monofacciale</name>
    <description>passaggio Peterlongo n.posti 10 - monofacciale</description>
    <ExtendedData>
        <SimpleData name="id">1</SimpleData>
        <SimpleData name="via">passaggio Peterlongo</SimpleData>
        <SimpleData name="zona">ZTL</SimpleData>
        <SimpleData name="tipologia">monofacciale</SimpleData>
        <SimpleData name="n_posti">10</SimpleData>
        <SimpleData name="descrizione">passaggio Peterlongo n.posti 10 - monofacciale</SimpleData>
    </ExtendedData>
    <LineString><coordinates>11.120116008895083,46.069743328815228
11.12009545743585,46.069829237795446</coordinates></LineString>
</Placemark>
```

**Listing 8: Excerpt of a KML file about bike racks in Trento**

---

[23] [http://www.comune.trento.it/Aree-tematiche/Cartografia/Download/Rastrelliere-per-biciclette](http://www.comune.trento.it/Aree-tematiche/Cartografia/Download/Rastrelliere-per-biciclette)
In order to accomplish the RDF conversion for this kind of data, we can exploit a feature of SPARQL to ease the mapping process: The data contains information about whether a bike rack is one or two sided. Here, “monofacciale” means one-sided, and using SPARQL/Update we can simply insert a mapping from the literal to a proper IRI. Subsequently, we can create a query over the input XML file and combine it with our newly defined mapping for “monofacciale”, as demonstrated in Listing 9.

```
INSERT DATA {
  lgd:type-single-sided rdfs:label "monofacciale" .
}

CONSTRUCT {
  ?s a lgd:BicycleParking ; rdfs:label ?l ;
  lgd:type ?t ; lgd:street ?st
} {
  <rastrelliere.kml> url:text ?src BINDSTRDT(?src, xsd:xml) AS ?xml
  ?xml xml:unnest("/*[local-name()='Placemark']" ?o)
  BIND(xml:path(?o,
    "//ExtendedData/SchemaData/SimpleData[@name='tipologia']") AS ?tn)
  OPTIONAL { ?t rdfs:label ?tn } # Map e.g. "monofacciale" to its proper type
  BIND(xml:path(?o, "name") AS ?l)
  BIND(xml:path(?o, "ExtendedData/SchemaData/SimpleData[@name='via']") AS ?st)
  BIND("http://qrowd-project.eu/resource/mt/bikerack-" AS ?ns)
  BIND(IRI(CONCAT(?ns, ENCODE_FOR_URI(?l))) AS ?s)
}
```

Listing 9: Declaration of mappings and referencing them in queries

### 4.3.2 Timetables for mode detection

The Trento dataset catalog includes time tables for trains\(^{24}\). Such information may serve as background knowledge to mode detection as it can be used as additional input to score the confidence of detected modes. For example, if according to the schedule data there is no train (and assuming completeness of the information) then detection of this mode is less likely to be the correct. In respect to RDF generation, this task involves zipping arrays by index. In Listing 10, we demonstrate how this is accomplished with Sparql-Integrate.

\(^{24}\) e.g. train timetable for Trento-Male [https://os.smartcommunitylab.it/core.mobility/timetable/10/555](https://os.smartcommunitylab.it/core.mobility/timetable/10/555)
INSERT DATA {
  eg:workload1 eg:workload """"{
    "stopIds": [ "TRENTO_STATION_FTM", "TRENTO_NORD" ],
    "stopNames": [ "Trento Staz.Ftm", "Trento Nord" ],
  }""""^^xsd:json
}

#CONSTRUCT { GRAPH ?x { ?s a eg:TrainStop ; rdfs:label ?l } }  
SELECT ?s ?l
WHERE {
  ?x eg:workload ?o .
  BIND(json:path(?o, "$stopIds") AS ?stops)
  BIND(json:path(?o, "$stopNames") AS ?stopNames)

  ?stops json:unnest (?stop ?i) .
  ?stopNames json:unnest (?l ?i) .

  BIND("http://qrowd-project.eu/resource/" AS ?ns)
  BIND(URI(CONCAT(?ns, 'stop-', ENCODE_FOR_URI(?stop))) AS ?s)
}

<table>
<thead>
<tr>
<th>s</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://.../stop-TRENTO_STATION_FTM">http://.../stop-TRENTO_STATION_FTM</a></td>
<td>&quot;Trento Staz.Ftm&quot;</td>
</tr>
<tr>
<td><a href="http://.../stop-TRENTO_NORD">http://.../stop-TRENTO_NORD</a></td>
<td>&quot;Trento Nord&quot;</td>
</tr>
</tbody>
</table>

Listing 10: Zipping arrays by index with Sparql-Integrate
5 Dataset Retrieval and Publishing with DCAT Suite

Figure 4: Overview of the DCAT Suite Components. The CKAN Importer converts metadata into the DCAT model. Subsequently, the DCAT Model can be used to deploy artifacts to other CKAN instances or - in the case of RDF - to triple stores.

The idea of the DCAT Suite is to offer a set of DCAT-centric tools and libraries for interacting with data portals, such as CKAN, and data stores, such as any triple store. Thereby, the DCAT vocabulary serves as the backbone for capturing knowledge about datasets and distributions. Its architecture is shown in Figure 4.

The main contribution of the DCAT Suite is a powerful data mapping engine that enables views of a CKAN model as if it was a DCAT-AP model based on bidirectional mappings. The envisioned advantage of this view-based approach is, that Linked Data tools, especially DCAT-AP based ones, are empowered to transparently read and write CKAN entities. This means, DCAT records can be readily converted to CKAN ones for publishing, and conversely, CKAN records can be readily converted to a proper semantic representation - provided that the mappings exist. Furthermore, DCAT Suite comes with a repository system for caching previously retrieved datasets or deploying local ones - similar to that of build tools used by developers.

DCAT Suite’s architecture comprises

- a Java library for the core functionality
- a command line wrapper enables standalone use and use in scripts, and
- a NiFi wrapper simplifies embedding the functionality in advanced workflows.

Note, that DCAT Suite and ATOS’s CKAN uploader25 share the functionality to interact with CKAN, such as uploading metadata and files. However, the difference is that DCAT Suite is built from the core on Linked Data technology (using the Apache

25 Cf. deliverable D4.2 ‘Data acquisition framework’
Jena (framework) and uses a DCAT(-AP) model for which a bidirectional mapper for CKAN is provided.

### 5.1 The DCAT Suite Command Line Tool

The command line tool offers convenient access to several fundamental tasks:

- Retrieve DCAT metadata and/or distributions from CKAN portals
- In the case of RDF distributions, load them into a triple store
- Publish metadata (and content) to CKAN portrals

The statement below demonstrates importing a metadata record from our QROWD data catalog, and using this as the base for bulk loading all suitable distributions into a triple store.

```bash
dcat import ckan \
--host=http://ckan.qrowd.aksw.org \
--dataset=org-linkedgeodata-osm-bremen-2018-04-04 > /tmp/dcat.nt

dcat deploy virtuoso --allowed=/writeable/dir/readable/by/virtuoso /tmp/dcat.nt
```

### 5.2 Bridging CKAN and DCAT-AP

The models for capturing information about datasets vary between CKAN and DCAT-AP: Although both models have many attributes in common, CKAN uses certain JSON representations that cannot be directly mapped to RDF using standard JSON-LD metadata. There exists a server-side CKAN plugin that defines an extensive CKAN-to-DCAT-AP mapping (~50 mappings). However, not every CKAN server has the plugin enabled and configured, and CKAN attributes outside of the mapping specification may not be exposed as RDF.

For this reason, we developed a complementary solution with the following features:

- Client-side solution which allows adding support for custom attributes as needed. For example, consider the `dcax:defaultGraphGroup` and `dcax:defaultGraph` attributes of the DCAT example in Listing 1.
- Bidirectional mapping system into which we integrated all mappings of CKAN's DCAT plugin
- View-based approach that gives consistent read/write access to DCAT entities using three APIs: Our DCAT API, Apache Jena’s RDF API and the used third-party CKAN API

Our approach comes with the following major advantages:

- Creation and modification of records about datasets and distributions can be done in a semantic way, and frees developers from dealing with the specifics of CKAN.
- The effort needed to add support for a new custom attribute is reduced to declaring a new mapping in a single place. For example; we exploit this mechanisms in order to annotate RDF distributions with the graph name that

---

26 [https://extensions.ckan.org extension/dcat/#rdf-dcat-to-ckan-dataset-mapping](https://extensions.ckan.org extension/dcat/#rdf-dcat-to-ckan-dataset-mapping)
should be used by default when loading the data into a triple store.

Naturally, bidirectional mappings also come with limitations.

- The main drawback is, that application interaction with the DCAT RDF layer will cause exceptions if operations are not supported by the underlying mapped CKAN model. For example, the CKAN-DCAT mapping allows only for one publisher record per dataset, so attempting to state more than that on the RDF level will fail.

As it turns out, the aforementioned CKAN-to-DCAT-AP mapping table documented at the CKAN-DCAT plugin’s Web page serves as good start, yet, additional information is needed to build a working implementation for it. Listing 11 shows an excerpt of our derived mapping specification crafted for this purpose and explained in the following.

```rml
[:dcatApMappingCollection
  a m:MappingCollection ;
  m:mapping [ m:target dcat:Dataset ; m:predicate dct:title ; m:key "title" ] ;
  m:mapping [ m:target dcat:Dataset ; m:predicate dct:description ;
    m:key "notes" ] ;
  m:mapping [ a m:CollectionMapping ; m:via "ckanTag" ; m:target dcat:Dataset ;
    m:predicate dcat:keyword ; m:key "tags" ] ;
  m:mapping [ a m:JsonArrayMapping ; m:type r2rml:IRI ; m:target dcat:Dataset ;
    m:predicate dcat:theme ; m:key "extra:theme" ] ;
(... 46 more mappings ...)
```

**Listing 11: CKAN-DCAT mapping specifications in RDF**

- **m:type**: The target RDF datatype values from a CKAN record should be converted to. May denote a numeric datatype, such as `xsd:integer`, but also `r2rml:IRI` in order to map a JSON string literal to an IRI in RDF. If omitted, `xsd:string` is assumed by default.
- **rdf:type**: The type of the mapping. First and foremost, it defines whether the mapped field is a single value or a collection. For collections, all items are subject to the mapping denoted by m:type.
  - **LiteralMapping**: The simplest type of mapping and assumed by default if no other mapping type is specified. Converts a single JSON field denoted by m:key to an RDF value of given type.
  - **JsonArrayMapping**: The CKAN-DCAT mapping specifies several instances where a CKAN field may hold a string serialization of a JSON array. This mapping types mediates between this representation and a set of triples derived from it - i.e. adding or removing a triple from the virtual graph generated from the mapping adds/removes the entry from string and vice version, respectively.
  - **CollectionMapping**: A mapping between a collection-valued field in the CKAN record and a set of triples in RDF. The `m:via` attribute references
the name of an internally registered converter. In our case, the CKAN Java model uses a CkanTag class to which tags on the RDF level (strings) need to be converted to.

- **target**: The DCAT entity type for which this mapping defines an attribute

### 5.2.2 DCAT Suite Command Line Client
The command line client enables carrying out tasks directly from the place many developers are already familiar with. The code snippet below demonstrates how to first download dataset metadata and distributions of a specific dataset from our QROWN CKAN instance using a given identifier and subsequently loading the datasets into a virtuoso triple store.

```bash
dcat import ckan \  
   --host=http://ckan.qrowd.aksw.org \  
   --dataset=org-linkedgeodata-osm-bremen-2018-04-04 > /tmp/dcat.nt

dcat deploy virtuoso \  
   --allowed=writeable/dir/readable/by/virtuoso /tmp/dcat.nt
```
6. Apache NiFi Integration

Sparql-Integrate and DCAT Suite have been packaged as Apache NiFi processors for use at the QROWN platform at https://nifi.qrowd.aksw.org/nifi/

Apache NiFi’s works on the basis of flow files, which are data records that associate (a pointer to) the payload/content with metadata. NiFi processors typically operate on one flow file at a time and thus consume and produce flow files.

The Sparql-Integrate processor is configured with a sequence of SPARQL queries or a reference to a .sparql file. As input, it receives flow files with the source data.

In order to reference the data of the flow file, the following convention is applied: The base URL, denoted by the symbol <> (technically an relative IRI that is an empty string), is interpreted as referring to the flow file. Hence, its content can be referenced in SPARQL using the pattern <> url:text?content. Following our defined practice (Section 4.1.1) of keeping the sequence of SPARQL queries for transformation separate from those that specify the workloads, the transformation can be readily reused in both NiFi and file system contexts. However, the workload specification has to be maintained separately for these environments. Figure 5 shows an example workflow of transforming source distributions to RDF and uploading them to CKAN using our processors.
7 RELATED WORK

7.1 Linked Data Generation

Several approaches have been proposed for the conversion of data to RDF. Triplify [1] is an early approach for exposing relational data as RDF. For every set of entities to be exposed, one has to specify a set of SQL queries that used special column naming conventions for crafting the identifiers and corresponding properties. An RDF-based mapping language following the same table-to-class, column-to-property principle is the D2RQ mapping language. The main limitation of these approaches is, that they require every predicate to be explicitly specified. The Relational data to RDF Mapping Language (R2RML) is an RDF-based language that overcomes these prior limitations. It introduces the notion of a Triples Map which comprises a logical table (i.e. a table/view name or an SQL query) and a set of Term Maps for creating RDF terms for the subject, predicate, object and optionally graph components from each row of the logical table. Our Sparqlification Mapping Language (SML) was originally developed with the goal to expose relational OpenStreetMap databases as virtual RDF graphs in the course of our LinkedGeoData community project. The SML syntax is inspired by the SQL CREATE VIEW statement, and we showed in a user study conducted in [2] that it is an easier-to-use alternative to R2RML. The CSV on the Web Working Group developed a set of related specifications and vocabularies for supplying metadata to CSV data. Applications can use this information for e.g. visualization, summarization (which essential information to display), conversion and validation. One of the first approaches for converting CSV to RDF using SPARQL extensions is TARQL. JARQL extends SPARQL by introducing a special namespace for properties that map to JSON attributes. This approach allows one to model basic graph patterns whose structure is akin to that of the JSON object to be matched.

Notably, according to documentation, JARQL neither supports unnesting of arrays nor retaining the order of arrays. Our approach - with the introduction of the json:unnest and xml:path property functions - mitigates both limitations.

SPARQL Generate [3] is a closely related effort that also builds upon Apache Jena extensions. However, in addition it introduces a new syntax for e.g. interaction which we address in a SPARQL-native way.

---

27 http://d2rq.org/d2rq-language
28 https://www.w3.org/TR/r2rml/
29 https://www.w3.org/TR/tabular-metadata/
30 https://github.com/tarql/tarql
31 https://github.com/linked-solutions/jarql
7.2 Data Catalogue Models

With respect to data catalogues, the DCAT vocabulary features the basic models for describing catalogues, datasets and distributions and has evolved into an established backbone for data portals. However, several applications require additional metadata. DataId [4] was developed for describing DBpedia [5] datasets and Linked Data ones in general. It builds on DCAT, defines several new terms and extensively reuses third party ontologies, such as DCTerms, VoID, PROV-O, SPARQL service description\textsuperscript{32}, ODRL, and FOAF. The DCAT Application Profile for data portals (DCAT-AP) also builds on DCAT, however many of its specified attributes differ from DataId. At present it is used by the European Data portal\textsuperscript{33,34} and around 20 other data portals across Europe\textsuperscript{35}.

8 CONCLUSIONS AND FUTURE WORK

In this deliverable, we first presented how SPARQL can be used for Linked Data generation from heterogeneous source data using our SPARQL-Integrate system. Datasets generated by this method and which also include appropriate DCAT metadata can be published to data portals (e.g. CKAN) and triple stores using our DCAT Suite system. Furthermore, we proposed best-practices for our tools based on concrete use cases. The Apache NiFi integration enables the use of our components in a popular workflow engine.

Our approach is also suitable for use within Big Data frameworks: Such frameworks operate by first partitioning the input data and then apply processing on each partition in parallel (typically across multiple computing nodes). Consequently, Linked Data generation using SPARQL Integrate queries can be parallelized if the input data can be appropriately partitioned.

\textsuperscript{32} http://www.w3.org/TR/sparql11-service-description/
\textsuperscript{33} https://www.europeandataportal.eu/
\textsuperscript{34} https://joinup.ec.europa.eu/release/dcat-ap-v11
(retrieved 2018-09-25)
REFERENCES


