OPAL
OPEN DATA PORTAL

Deliverable D4.2 Konvertierungskomponente

Autoren: Afshin Amini, Zafar Habeed Syed, Matthias Wauer

Veröffentlichung	Vertraulich
Fälligkeitsdatum	31.12.2018
Fertigstellung	08.02.2019
Arbeitspaket	AP4
Typ	Software
Status	Final
Version	1.0

Kurzfassung:

In OPAL werden Metadaten zu offenen Datensätzen aus verschiedensten Quellen gesammelt. Hierfür wurde in D4.1 bereits ein Vokabular definiert, mit dem die unterschiedlichen Quelldaten in RDF repräsentiert werden können. Mit Hilfe der Konvertierungskomponente wird die Transformation aus den Quellformaten (z.B. HTTP, JSON oder RDF) in das Zielformat RDF umgesetzt. Dieses Deliverable beschreibt die erste Version dieser Komponente, die sich aus einem entsprechenden Modul in der Crawler-Komponente Squirrel sowie einem separaten Dienst zur Übersetzung in das OPAL-Vokabular zusammensetzt.

Schlagworte:

Konvertierung, Transformation, RDF, Vokabular

mFUND
Das Startkapital für die Mobilität 4.0

Bundesministerium
für Verkehr und
digitale Infrastruktur

Inhalt

1 Introduction

OPAL collects and provides metadata from different open data portals. Consequently, the available metadata coming in from the crawling component can vary regarding its representation. In order to provide a consistent view on this metadata and allow further processing, such as interlinking and fusion, OPAL needs to transform the incoming data into a homogeneous representation.
In Deliverable D1.3 we have designed a high-level architecture of the OPAL system, which includes a transformation layer which is supposed to provide a data conversion framework to execute this task. In this deliverable we discuss the functionality and design of this framework. In addition to that, we describe the current implementation and example use cases for selected sources.
The work package description of AP4 also suggested that the conversion component should support providing the OPAL metadata in different representations, such as Turtle and JSON-LD, depending on the requirements of the APIs and front-end applications. Such different representations could already be provided by available functionality of common triple stores used for data storage. Thus, it is currently not necessary to define this functionality for the conversion component. Depending on further research, if there is a need to transform the OPAL metadata into different vocabularies (e.g., different DCAT-AP profiles for certain countries), such functionality may be added in a later iteration of this deliverable.

Note: Throughout the document, we use the following defined prefixes to represent RDF properties and resources:

Prefix	**URI**
xsd	<http://www.w3.org/2001/XMLSchema#>
dct	<http://purl.org/dc/terms>
foaf	<http://xmlns.com/foaf/0.1>
dcat	<http://www.w3.org/ns/dcat#>
rdfs	<http://www.w3.org/2000/01/rdf-schema#>
skos	<http://www.w3.org/2004/02/skos/core#>
vcard	<http://www.w3.org/2006/vcard/ns#>
opal	<http://projekt-opal.de>

2 RDF Transformation of Open Data Metadata

In this section, we provide detailed information about the metadata representation in OPAL. To represent the metadata information, we use existing DCAT - a vocabulary specification to store and publish metadata information about catalogs, datasets and distributions in the form of RDF description. The crawler component query/crawl the metadata information from seed catalogs. This information is categorized as follows:

1. **Catalog:** This includes information pertaining to the seed catalog. For example, the homepage of the catalog, publishers, the list of datasets published in the catalog etc. Each catalog is assigned a URI. These URI's are generated by appending the unique identifier (provided by the crawler) of the catalogs to the base URI <http://projekt-opal.de/catalog>. Note that, all the catalogs (URI's) are represented as the type (rdf:type) of dcat:Catalog -- a DCAT specification. Additionally, for each catalog, all the metadata information of that catalog is represented using the following RDF properties,

Metadata	**RDF-Property**	**Range**
Title	dct:title	xsd:String
Description	dct:description	xsd:String
Language	dct:language	dct:LinguisticSystem
Homepage	foaf:homepage	foaf:Document
Publisher	dct:publisher	foaf:Agent
Geo-Location	dct:spatial	dct:Location
Issued	dct:issued	xsd:dateTime
Modified	dct:modified	xsd:dateTime
License	dct:license	URL
Dataset	dcat:dataset	dcat:Dataset
Format	dct:format	dct:MediaTypeOrExtent

Table 1. RDF representation of Catalog

2. **Dataset:** All the datasets belonging to a catalog are represented as a list of dataset URI's using the RDF property dcat:dataset. The dataset URI's are generated by appending the unique identifier of the dataset and the base URI opal:dataset and assigned the type information, the dcat:Dataset. All the metadata information of these datasets such as their available distributions, last modification date etc., are represented against the corresponding subject (Dataset) URI by the following RDF properties

Metadata	**RDF-Property**	**Range**
Title	dct:title	xsd:String

Description	dct:description	xsd:String
Date Issued	dct:issued	xsd:dateTime
Date Modified	dct:modified	xsd:dateTime
Language	dct:language	dct:LinguisticSystem
Publisher	dct:publisher	foaf:Agent
Creator	dct:creator	foaf:Agent
Accrual Period	dct:accrualPeriodicity	dct:Frequency
Geo-Location	dct:spatial	dct:Location
Temporal Period	dct:temporal	dct:PeriodOfTime
License	dct:license	URL
Category	dcat:theme	skos:Concept
Contact Point	dcat:contactPoint	vcard:Organization
Landing Page	dcat:landingPage	foaf:Document
Distribution	dcat:distribution	dcat:Distribution

Table 2. RDF representation of Dataset

3. **Distribution:** Similar to the catalogs and datasets each distribution is assigned a URI by appending the unique identifier of the distribution (provided by squirrel) and the base URI opal:distribution. Additionally, the incoming metadata information of the distributions from squirrel is extracted/queried and is represented as values of the following RDF properties

Metadata	**RDF-Property**	**Range**
Title	dct:title	xsd:String
Description	dct:description	xsd:String
Date Issued	dct:issued	xsd:dateTime
Date Modified	dct:modified	xsd:dateTime
License	dct:license	URL
Access URL	dcat:accessURL	rdfs:Resource
Download URL	dcat:downloadURL	rdfs:Resource

Table 3: RDF representation of Distributions

3 Conversion Component

Figure 3.1 depicts the conversion component. Prior to converting the datasets into homogeneous representation, all the metadata from the input portals, also called seed catalogs, is extracted/queried and stored in the form RDF triples (see 3.a in figure). The representation of collected data is heterogeneous in nature depending on the vocabulary specification used, for example mcloud, govdata etc.

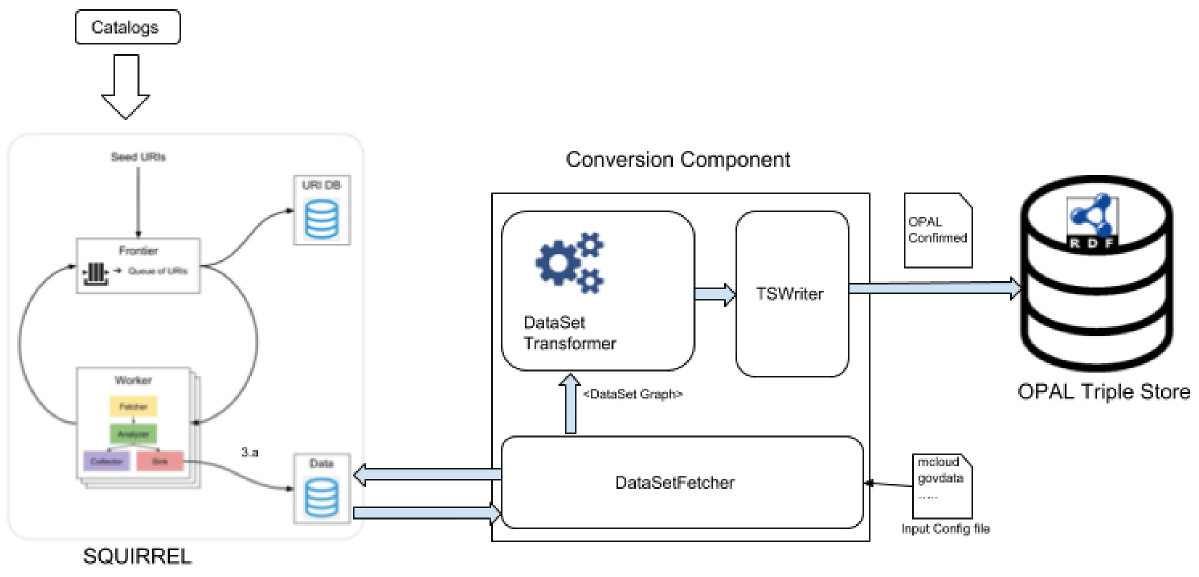

Fig 3.1. Conversion component diagram

3.1 Transformation into RDF

Squirrel has two components (Analyzer, Sink) for transforming data from different sources to RDF and store them in a persisting media.

3.1.1 Analyzer

Analyses the fetched data and extract triples from it, and also it can analyse data from different input formats (RDF, HTML, HDT). The respective components are as follows:

- RDFAnalyzer - Analyses RDF formats.
- HTMLScraperAnalyzer - Analyses and scrapes HTML data base on Jsoup selector-syntax.
- HDTAnalyzer - Analyses HDT binary RDF format.

Depending on the type, the Analyzer takes as input a configuration file. The configuration files contain mappings of data or HTML fields and their corresponding RDF properties. A list of configuration files are here. The Analyzer component generates a unique identifier for each dataset. The extracted data is stored against the identifier in the form of RDF triples using the RDF properties from the configuration files.

3.1.2 Sink

Responsible for persisting the collected RDF data, and can be one of the follows:

- *FileBasedSink* - persists the triples in NT files,
- *InMemorySink* - persists the triples only in memory, not in disk (mainly used for testing purposes).
- *HdtBasedSink* - persists the triples in a HDT file (compressed RDF format - http://www.rdfhdt.org/).
- *SparqlBasedSink* - persists the triples in a SparqlEndPoint.

3.2 Transformation into the OPAL vocabulary

As shown in Figure 3.1, the conversion component takes as input the metadata information from various catalogs collected and published by the Squirrel and converts it into homogeneous representation based on the OPAL specifications described in Section 2. Initially, the conversion component is provided with the list of seed catalogs. For each catalog, the conversion component query all the metadata information of the catalog provided by the Squirrel. A URI is generated for each catalog and all the metadata information related to the catalog is stored against the generated URI. After a catalog is created all the datasets belonging to the catalog are transformed into OPAL specification and all the information stored in an RDF store. This process is carried out in the following three stages,

Gathering Dataset Information: To do this, the conversion component queries the list of datasets for each catalog (see DataSetFetcher module in Fig 3.1). For each dataset, an RDF graph is constructed by querying all the metadata information of the dataset and its distributions. Note that, at this stage the dataset contain information according to the vocabulary specifications of the source catalog. After the graph is constructed, it is then passed to Converter component for further processing.

Transform Dataset Information: The converter module takes as input the dataset graph generated by the DataSetFetcher and generates a new dataset graph containing the information as per OPAL vocabulary specifications. As a first step, a URI is generated for the new dataset by querying the unique identifier of the input dataset graph and appending it to the base URI for the dataset as described in Section 2. All the metadata information from the input dataset graph is extracted and stored against the newly generated URI using the RDF properties described in Section 2. Similar to the dataset, new URI's are generated for each distribution of the dataset by querying the identifier of distributions from the input dataset graph. All the metadata information of distributions are stored against their corresponding newly generated URI's. All the URI's of distributions are linked and stored against the dataset URI using the RDF property, dcat:distribution to form the new dataset graph.

Writing to Triple Store: The triple store writer (see TSWriter in figure) module takes as input the transformed datasets generated by the converter. All the datasets collected are stored in an RDF store. In OPAL, we use Apache Fuseki to store and query metadata of the datasets.

4 Implementation

In OPAL, the implementation of the conversion component is carried out using Spring Boot -- an open-source Java based framework used to create stand-alone spring applications that are ready to be run. The application can be started by providing minimal configuration information. The configuration includes the list of input catalogs, information source RDF triple store containing the crawled information by Squirrel and the target RDF triple for writing the converted datasets. In its current form, the DatasetFetcher module is designed to run sequentially, i.e., one catalog at a time. This process will be parallelized in the future releases. The converter module is implemented to handle conversion of datasets parallely. Finally, the TSWriter module carries out the writing operation in batches where each batch contain a collection of pre-defined number of datasets. The size of batch can be easily configured based on the available resources or the size of catalog. The component is open-source and is available at https://github.com/projekt-opal/convertion .

5 Conclusions

In this deliverable document, we described the working and implementation of conversion component. The conversion component transforms datasets crawled and collected by the Squirrel from heterogeneous sources into uniform representation. To do this, the conversion component make use of existing vocabularies. This document present in detail the vocabularies used in OPAL.

The first version of conversion component is implemented in Java and bundled into a docker image. The process of transforming incoming datasets is partially parallel. In the future releases, we will focus on enhancements and to fully make the process parallel.