OPAL
OPEN DATA PORTAL

Deliverable D8.2
Erster Portalprototyp

Autoren: Zafar Habeeb Syed, Afshin Amini, Adrian Wilke, Matthias Wauer

Veröffentlichung	Vertraulich
Fälligkeitsdatum	31.12.2018
Fertigstellung	31.01.2019
Arbeitspaket	AP8
Typ	Software
Status	Final
Version	1.0

Kurzfassung:
Das OPAL-Projekt entwickelt einen Prototyp für ein umfassendes Open-Data-Portal mit Fokus auf Metadaten. In diesem Bericht wird die erste Version dieses Prototyps vorgestellt. Darin werden die einzelnen Komponenten des Prototyps und deren Komposition beschrieben. Außerdem wird auf die gegenwärtigen Daten- und Kontrollflüsse eingegangen.

Schlagworte:
OPAL, Portal, Prototyp, Komponenten

mFUND
Das Startkapital für die Mobilität 4.0

Bundesministerium
für Verkehr und
digitale Infrastruktur

Inhalt

1 Introduction

In OPAL work package 8 we aim to integrate the developed components of OPAL into an extensible component-based Web application. According to the description of work, this task should conform to the architecture described in Deliverable D1.3 (see Figure 1).

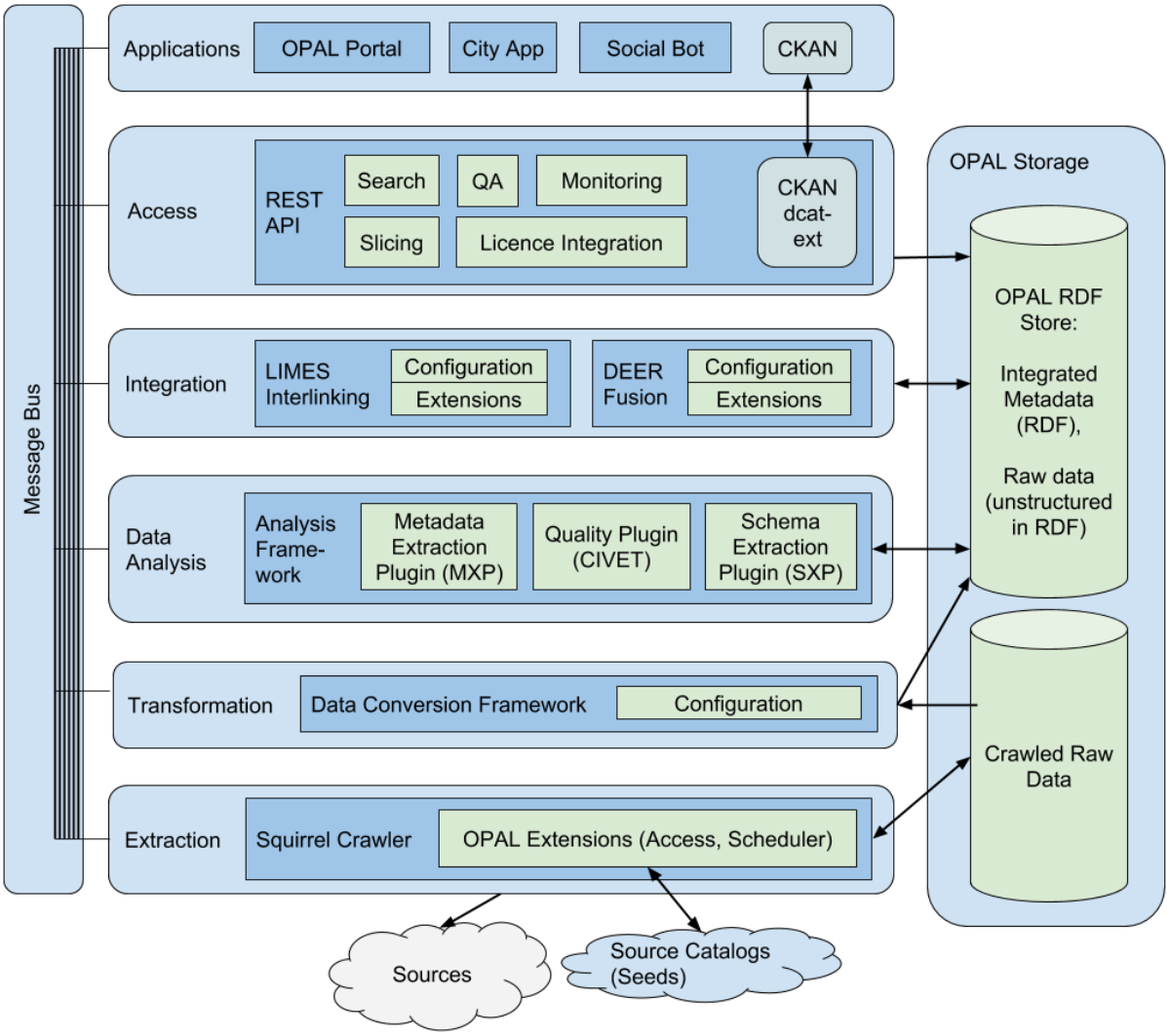

Figure 1: OPAL Architecture (from D1.3)

Consequently, in this Deliverable we describe the first version of the portal prototype. The document is structured as follows: All portal components are described individually in Section 2, including introductions of components that are work in progress. In Section 3 we discuss how these components are integrated into the portal, including the data flow, control flow, and component composition.

2 Description of the Portal Components

The OPAL platform has been designed as a layered architecture. In the following, a short description of layers is given.

2.1 Extraction

This layer contains the functionality for focused crawling of Open Data sites. The Squirrel crawler has been developed for the extraction of metadata from Open Data portals, for which it has been extended with OPAL-specific extensions for accessing specific sources efficiently and also scheduling the crawling process as necessary.

2.1.1 Squirrel

Squirrel core is divided into two main components: frontier and worker. The execution of Squirrel requires one frontier running, while the user can set how many workers the system supports. The frontier is initialized by a list of input seeds. It will add all the identified URI's to a queue and to a filter. Once the frontier receives a call from workers, will give a subset of the URI's in the queue to the workers.

The worker will only be initialized if there is a frontier available to connect to. Initially, it will request new URI's to crawl to the frontier. Then, it will fetch data available from the URI. After fetching, it will analyze the fetched file to extract triples from it and thus, store data in a sink. In the end, all the URI's found by the analyzer will be serialized and sent to the frontier. The frontier receives these new URI's, checks if they are present in the filter and add to the queue only the ones that are not.

The frontier also registers the IP number of the URI and assigns that IP to the first worker that requests it. By doing that, a worker will be responsible for crawling URI's from the same IP number.

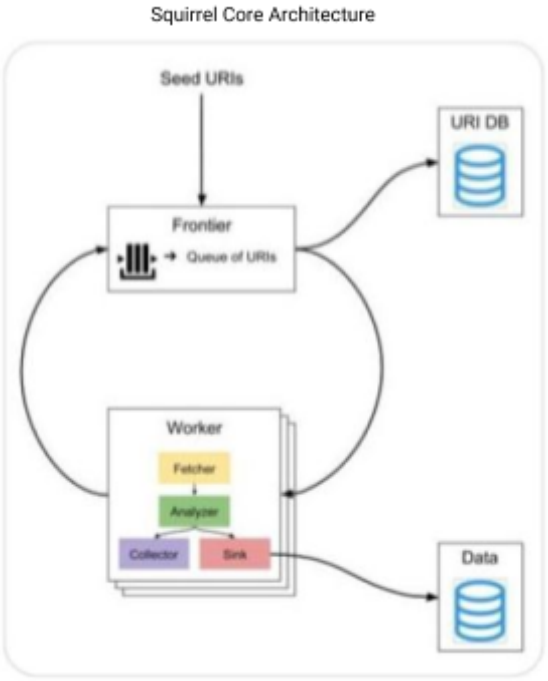

Figure 2: Architecture of the Squirrel crawling framework

2.2 Transformation

In this step, all URIs, every resource, and every property are adjusting to becoming OPAL vocabulary confirmed (see Deliverable D4.1). For this, a conversion tool is designed that in parallel for each dataset fetches all distributions, publisher, properties, and objects; then, does the conversion of them(see Deliverable D4.2). after that, for each data set CIVET component is

used to calculate quality metrics for it (see Deliverable D3.1). Finally, it stores the result in a TripleStore.

2.3 Data Analysis

When raw information from a Web site or API has been extracted, a data analysis framework takes care of three aspects:

1. Further metadata extraction (e.g., analyzing the description text of the dataset and linking resources like licenses in a knowledge base) via the MXP component. A first version will be developed for Deliverable D3.3.
2. Measuring the metadata quality with metrics to be defined in Deliverable D3.1 via the CIVET component, described in Deliverable D3.2.
3. Attempting to extract schema information for well-known content types via the SXP component (in collaboration with the LIMBO project, a first version is to be developed for Deliverable D3.4).

2.4 Integration

The data integration layer consists of two components. The interlinking component, based on the LIMES framework, attempts to determine *identity sets* of crawled resources describing the same dataset (typically from different sources), as well as related datasets (based on different features). The fusion component, using the DEER framework, is applied for merging the potentially conflicting metadata items from different sources into a coherent representation for a single dataset. Both components are available, but their integration in the portal has not been executed yet.

2.5 Access

The data access layer contains the REST API, which consists of

1. OPAL-specific services for searching, monitoring, question answering (QA), dataset slicing and license integration. CIVET for quality measurement is now available via CLI, and also java importing package. The other components are not available as a service in the current version.
2. the CKAN API as a common de-facto standard for accessing metadata, aiming to enable easy reusability of the integrated enhanced metadata.

The CKAN API is actually provided as part of the CKAN installation in the application layer. The API interface to the OPAL architecture is enabled by a service which will be based on the CKANext-DCAT plugin, which allows converting between the DCAT RDF formats and the internal relational metadata representation in CKAN, and also CKANext-Harvest which is used to provide a common harvesting framework for CKAN extensions and adds a CLI and a WUI to CKAN to manage harvesting sources and jobs. In order to provide datasets to be harvested by CKANext-Harvest a TripleStore dumper has been implemented that dumps RDFs in a TripleStore into paginated files to harvest them page by page (not facing memory problems) in CKAN, and also a file server is designed that make the paginated files available for CKANext-Harvest. For CKAN be able displays quality metrics appropriate RDF-Profiling is set in the extended parser and serializer classes.

2.6 Applications

The application layer provides four distinct services:

1. The OPAL portal is the prototype implementation of the open data portal, including comprehensive search functionality over the integrated metadata and visualizations, e.g., for quality metrics. Designing and implementation of the OPAL portal have been started and the first prototype will be delivered by Deliverable D8.3.
2. The city app is a mobile application for finding relevant datasets, primarily based on the search API. The development will start now, to be finished for Deliverable D7.3.
3. The social bot suggests datasets in certain social platforms (e.g. Twitter) using the QA API in the access layer. The development will start now, to be finished for Deliverable D7.4.
4. CKAN provides the integrated metadata of OPAL in a traditional CKAN portal Web interface, primarily to enable a preliminary user interface and easy reuse of existing CKAN connectors for accessing the OPAL data.

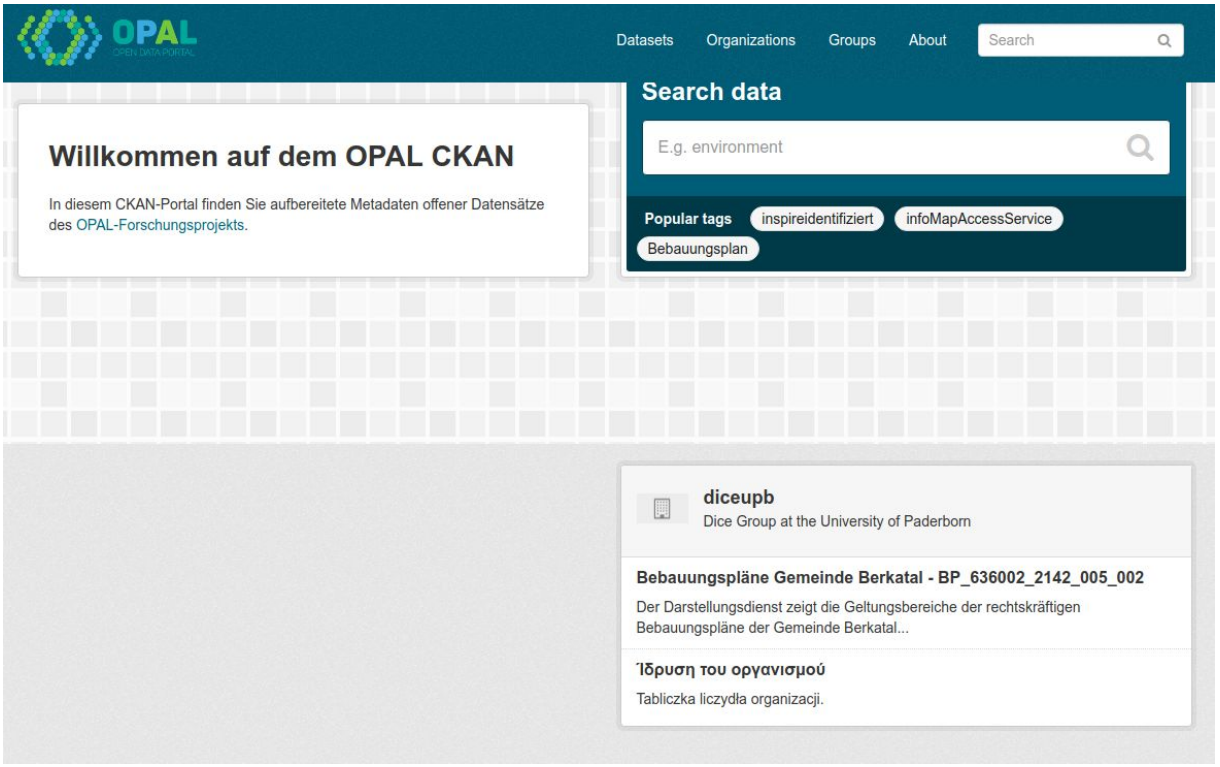

Figure 3: Screenshot of the CKAN user interface of the first OPAL portal prototype.

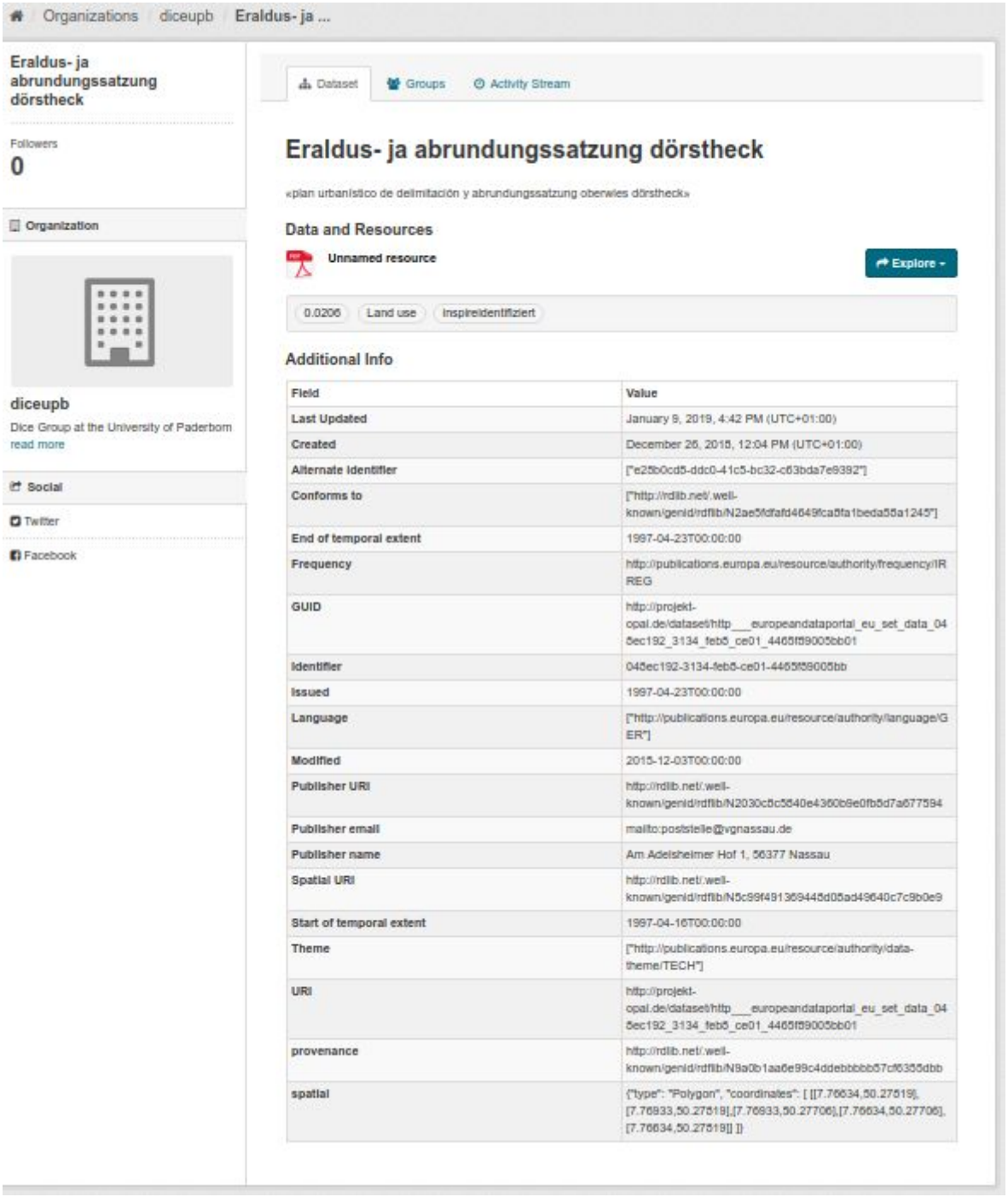

Figure 4: Screenshot of dataset details in the first version of the CKAN UI in OPAL.

2.7 Storage

OPAL data is stored in two distinct areas. The Squirrel crawler stores downloaded documents in a TripleStore (Apache Fuseki in this version) with a different endpoint for different portals. All other data, including the quality metrics and all metadata (OPAL vocabulary confirmed), are managed as RDF in another triple store (Apache Fuseki in this version). Both data are available via an SPARQL endpoint. In addition, metadata of datasets (including quality metrics) is also available in CKAN, in which the user can search, filter and also browse the data set.

3 Integration of the Portal

In this section we discuss how the previously described components interact in the portal prototype.

3.1 Data Flow

In the current version of OPAL, the data flow is as shown in the following Figure 1. First, Crawler harvests data from the portals and URLs, and store it in a TripleStore, then conversion component makes sure entities are OPAL-Vocabulary confirmed, then pass those data to CIVET, after calculating quality measurements the RDF results are stored in a TripleStore. Then, TripleStore Dumper exports all RDFs in paginated files. Finally, CKANext-Harvest collects all files and stores their metadata in its own database.

Note that in the first four steps data is not passed directly, but managed in a triple store, as detailed in Section 3.3. As such, the default interface between these components is SPARQL.

Figure 5: Data flow in the first OPAL portal prototype (simplified).

3.2 Docker Composition

The first prototype of OPAL has been implemented using Docker Compose in order to orchestrate the different components, as specified in Deliverable D8.1. The overview of the docker-compose file is as shown in Listing 1.

```
version: '2'
services:
 ckan-db:
  image: postgres:9.6.0
  container_name: ckan-hobbit-db
  environment:
   - ...
  volumes:
   - ...
  restart: always
 ckan-solr:
  image: earthquakesan/ckan-solr:2.8.0
  container_name: ckan-hobbit-solr
  restart: always
 ckan-redis:
  image: redis:4.0.11
  restart: always
 ckan:
  image: mwauer/ckan:2.8.1-opal
  environment:
```

```
    - ...
  volumes:
    - ...
  restart: always
  ports:
    - ...
fileStorage:
  image: busybox
  volumes:
    - /files
fileServer:
  image: mnafshin/simple_file_server
  volumes_from:
    - fileStorage:ro
  restart: always
  ports:
    - ...
fuseki-data:
  image: busybox
  volumes:
    - /fuseki
civetTripleStore:
  image: stain/jena-fuseki
  volumes_from:
    - fuseki-data:rw
  ports:
    - ...
  environment:
    - ...
conversionTool:
  image: mnafshin/conversion-tool
  environment:
    - ...
  ports:
    - ...
tripleStoreDumper:
  image: mnafshin/triple-store-dumper
  volumes_from:
    - fileStorage:rw
  environment:
    - ...
  ports:
    - ...
```

Listing 1: Docker Compose configuration of the first OPAL prototype

The following services are included in this composition:

- **ckan-db** is a postgres database container that stores CKAN's data (like user credentials, metadata of harvested datasets, harvesting sources, etc.).
- **ckan-solr** is an Apache Solr container which is used in CKAN as an internal text search engine.
- **ckan-redis** is used as a queue (Message Broker) for harvesting datasets in CKANext-Harvest plugin. It is used to bring up gather_consumer and fetch_consumer queues.
- **ckan** is the CKAN instance itself.
- **fileStorage** is used to share files between tripleStoreDumber and fileServer.
- **fileServer** is used to make files available for CKANext-Harvest plugin.
- **fuseki-data** is used to store Apache Fuseki data permanently.
- **civetTripleStore** is an Apache Fuseki container that store RDFs after converting by conversion-tool and extended by CIVET.
- **conversionTool** is the conversion-tool service that also runs CIVET after converting each dataset's entities.
- **tripleStoreDumper** is the TripleStore Dumper service that dumps civetTripleStore in fileStorage.

3.3 Control Flow / Invocation

In this version of the OPAL controlling flow between different components is triggered manually. First Squirrel starts crawling and storing in a TripleStore. Then, conversion-tool is triggered to make the entities OPAL-Vocabulary confirmed and quality metrics are calculated for each dataset, and the results are stored in a TripleStore. After that, tripleStore Dumper is called to store all RDFs in the previous step TripleStore in paginated files. Finally, CKANext-Harvest is called for each file which is generated in the previous step, i.e. for each file, a harvesting source, and a job is created, respectively.

4 Conclusions

In this deliverable, we have described the first version of the OPAL portal prototype. For this version, around 800K datasets and their distributions have been harvested and are available in the CKAN portal, which provides the initial user interface and an API for programmatic access to the linked data.

Next, we will extend the prototype by automating the interaction between the components via the message bus, such that newly crawled information will be automatically processed and updated.