OPAL
OPEN DATA PORTAL

Deliverable D1.3
Architektur

Autoren: Matthias Wauer, Ivan Ermilov

Veröffentlichung	Vertraulich
Fälligkeitsdatum	31.12.2017
Fertigstellung	30.06.2018
Arbeitspaket	AP1
Typ	Bericht
Status	Final
Version	1.0

Kurzfassung:
Dieser Bericht beschreibt den Entwurf der Architektur für OPAL. Auf der Grundlage der erforderlichen Funktionalität (Deliverable D1.1) definiert das Dokument die notwendigen Komponenten und spezifiziert die Schnittstellen zwischen den einzelnen Architekturschichten.

Schlagworte:
Architektur, Schichten, Komponenten, Dienste, Schnittstellen

mFUND
Das Startkapital für die Mobilität 4.0

Bundesministerium
für Verkehr und
digitale Infrastruktur

Inhalt

1 Introduction

The primary goal of the OPAL project is to enable easier access to the Open Data provided in German portals. This requires a range of functionalities for crawling, extracting, analyzing, converting, integrating and using the respective metadata. In Deliverable D1.1 we summarized the requirements towards a platform providing these functionalities. In order to provide a foundation for the technical implementation of the OPAL portal, this deliverable will describe the OPAL architecture, which was designed in the past months with the following goals in mind:

- Definition of all major components and services required in the project
- Specification of the primary interfaces between the components
- Modular design of respective components, so they can be reused in different contexts
- Loose coupling, so the components can be developed and evolved independently
- Extensibility of the system, particularly with regards to additional sources, functionality and metadata elements
- Independence of deployment scenarios/computing infrastructures

2 OPAL Architecture

The OPAL platform has been designed as a layered architecture. This section attempts to give an overview of the architecture, describing the primary functionality of each layer.

2.1 Architecture Overview

A high-level overview of the OPAL architecture is shown in Figure 1. It is separated into the following layers:

1. The **extraction** layer contains the functionality for focused crawling of Open Data sites.
2. The **data analysis** layer contains an extensible framework with components for extracting certain metadata elements, processing quality metrics, and extracting schema information for a dataset.
3. The **transformation** layer translates the different metadata representations into a common vocabulary.
4. The **integration** layer links and fuses the different metadata descriptions from different portals for each dataset, and computes relationships between different datasets.
5. The **access** layer provides an API for programmatic access to the portal functionality.
6. The **applications** layer contains the end-user applications.

In addition to these layers, a vertical **storage** handles raw data related to the crawling framework, as well as different metadata representations in RDF. The vertical **message bus** handles event and message propagation between the layers.

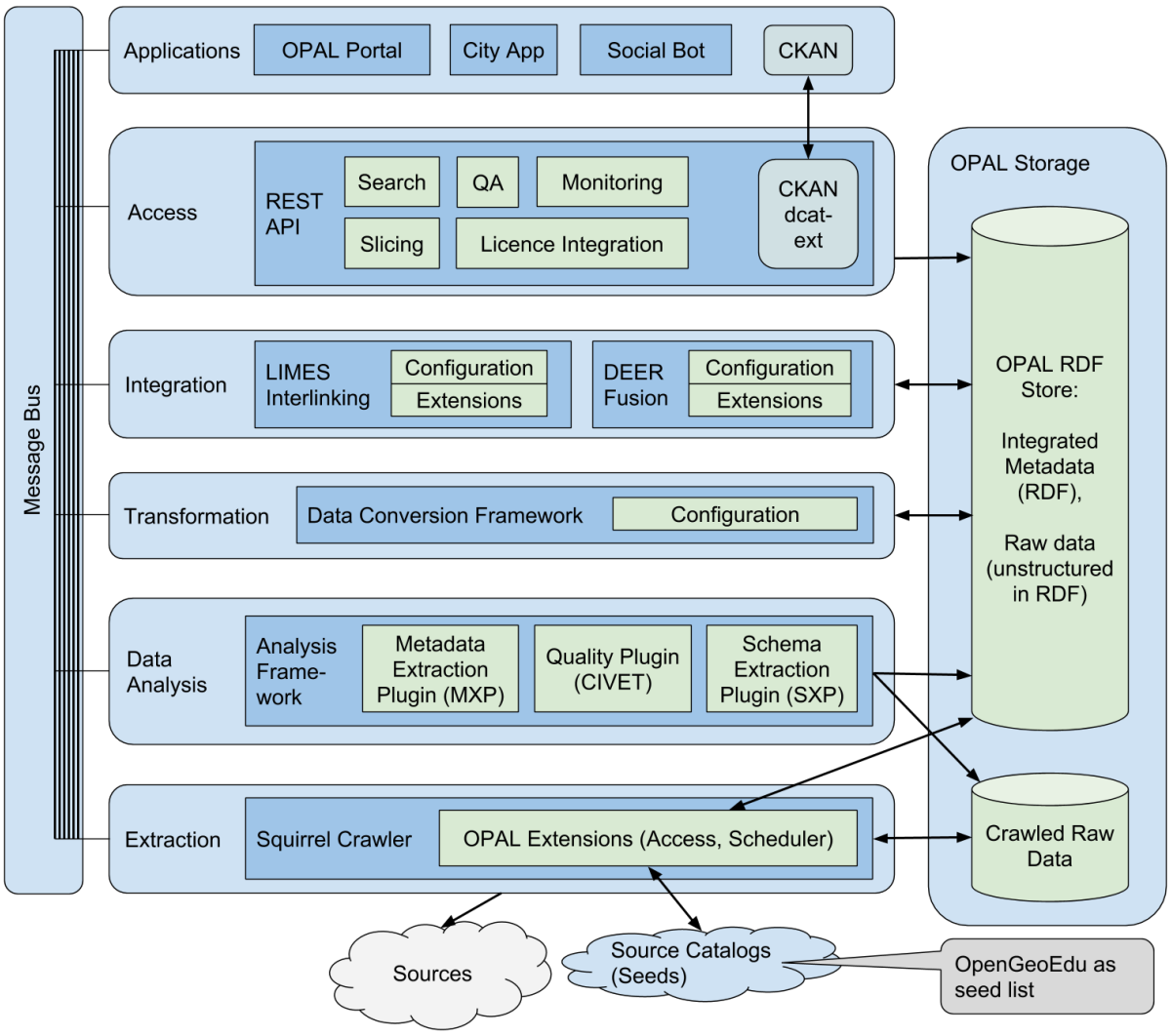

Figure 1: OPAL Architecture

2.2 Component Descriptions

The following sections will describe the functionality of each component.

2.2.1 Extraction

The extraction of metadata from Open Data portals is done by focused crawling. We have selected the Squirrel **crawler**, which we will extend with OPAL-specific extensions for **accessing** specific sources efficiently and **scheduling** the crawling process as necessary.

The crawler operates on sources, which usually are Web pages, but can also include APIs of common tools like CKAN to be more effective. The crawler will start with a source catalog containing so-called seeds for the crawling process. In addition to the portals listed in D1.1, we will use the subset of the datasets listed on OpenGeoEdu[1] relevant for Germany to complement the initial seed list.

The crawler is being implemented in OPAL work package 2. The initial specification will be documented in OPAL Deliverable D2.1.

[1] https://portal.opengeoedu.de/

2.2.2 Data Analysis

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

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

All of these components are implemented as extensions of the framework. The metadata extraction plugin (MXP) aims to extract metadata items from semi-structured formats, such as tables, as well as unstructured sources like description texts, and align it to respective properties in extensible vocabularies. We plan to reuse aspects of the TAIPAN, AGDISTIS and FOX tools for this plugin.

The quality metrics are computed by the CIVET component. CIVET processes metadata transformed in a common format by the conversion component (see next section).

The schema extraction plugin (SXP) can identify the dataset topic and schema in two ways: by identifying topical facets in textual descriptions and other metadata features, and using the actual data. For the second case, we depend on the transformation of different source formats into a common input format, provided by task 4.2 of the LIMBO project.

2.2.3 Transformation

The data conversion framework transforms the source-specific generic metadata into a common vocabulary and respective representations of specific metadata elements, e.g., in DCAT-AP and OPAL-specific extensions (to be defined in D4.1 vocabulary specification). Furthermore, it generates the index structures necessary of effective search functionality.

2.2.4 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.

2.2.5 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 licence integration, as well as
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 dcat-ext CKAN plugin (or a similar approach), which allows to convert between the DCAT RDF formats and the internal relational metadata representation in CKAN.

2.2.6 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 visualisations, e.g., for quality metrics.
2. The city app is a mobile application for finding relevant datasets, primarily based on the search API.
3. The social bot suggests datasets in certain social platforms (e.g. Twitter) using the QA API in the access layer.
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.

2.2.7 Storage

OPAL data is stored in two distinct areas. The Squirrel crawler stores downloaded documents in a raw data container (first implementation is file based). All other data, including the quality metrics and all metadata, are managed as RDF in a triple store. Data from different layers and components will be handled respectively, i.e., in different graphs.

3 Interfaces

All the layers store their data in the OPAL storage layer, which is accessible by other components. For example, crawled RDF data from extraction layer can be accessed by the data analysis components and the transformation framework. To enable communication between the components we introduce message bus layer based on AMQP protocol. The components can exchange messages such as: crawling done (the data is stored in graph A), quality assessment done (the results are stored in graph B) etc. In such an architecture the components can listen to the messages and start processing when necessary.
The particular API specifications are defined in the resp. Github repositories.

4 Conclusion

In this deliverable we specified the general architecture of OPAL.