A Workflow for generation of LDP

Noorani Bakerally, Antoine Zimmermann and Olivier Boissier

Univ Lyon, IMT Mines
Saint-Étienne, CNRS, Laboratoire Hubert Curien UMR 5516, F-42023 Saint-Étienne, France
{noorani.bakerally,antoine.zimmermann,olivier.boissier}@emse.fr

Abstract. Linked Data Platform 1.0 (LDP) is the W3C Recommendation for exposing linked data in a RESTful manner. While several implementations of the LDP standard exist, deploying an LDP is still manual and tighly coupled to the chosen implementation. As a consequence, the same design (in terms of how the data is organised) is difficult to reuse in different LDP deployments. In this paper we propose a workflow for LDP generation to automatize the generation of LDPs from static, dynamic and heterogeneous data sources while keeping the design loosely coupled from the implementation.

Keywords: RDF, Linked Data, Linked Data Platform

1 Introduction

The aim of the Linked Data Platform (LDP) 1.0 W3C Recommendation [7] is to standardize RESTful access to RDF data. Linked data platforms complying with the LDP standard, which we refer to as LDPs, can be useful in different contexts such as open data, where there is need to have a homogeneous view and access to data to facilitate their exploitation.

Currently, a number of LDP implementations exist. They are mostly referenced in the standard conformance report[1]. Yet, deploying an LDP from existing data sources is still complex. Currently, it requires the development of LDP generators to transform data resources from their native structures to LDP resources which can be deployed in LDP stores. Moreover, while doing so, it is possible that design decision related to the final platform are hardcoded in the LDP generator enhancing a tight coupling between the design and implementation complexifying both the maintainability and reusability. In summary, current LDP implementations are in their early stages as there is little to no support for automating the generation and deployment of LDPs from existing data, even if it is already in RDF.

In our previous works, we described an approach [1] for the generation of LDPs and a simple proof of concept [3] of doing so from static and heterogeneous data sources using existing LDP implementations. In this paper, our objective is to present the extension of the former proof of concept as a refined approach that we refer to as the LDP generation workflow. In addition to considering static and heterogeneous data sources, the novelty of this approach is the ability to deploy LDPs on dynamic data sources and to provide fresh data for LDP resources at query time, a feature which is not yet natively supported

[1] https://www.w3.org/2012/ldp/hg/tests/reports/ldp.html on 11 March 2018

directly by any existing LDP implementation. To this end, first we provide an overview of our approach (Sec. 2) followed by its implementation (Sec. 3) and demonstration in different scenarios (Sec. 4). Finally, we conclude with an outlook on future works (Sec. 5).

2 Our Approach: The LDP Generation Workflow

Our approach is based on model-driven engineering that involves using models as first-class entities and transforming them into running systems by using generators or by dynamically interpreting the models at run-time [5]. Doing so enables separation of concerns thus guaranteeing higher reusability of systems' models [8].

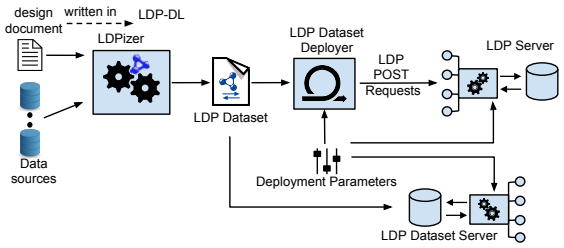

Fig. 1: General overview of the LDP Generation Workflow

Fig. 1 shows a general overview of the approach that includes two processes: LDPization and deployment. In the former process, the LDPizer consumes a design document written in our language, LDP-DL, that we use as a domain-specific language, a core component of model-driven engineering, to explicitly describe LDP design models. Concerning LDP-DL, its abstract syntax is described in our technical report [4] while its concrete RDF syntax of LDP-DL is given in its specification [2]. The LDPizer interprets the model and exploits the data sources to generate what we call an LDP dataset, which is a structure to store LDP resources introduced to abstract ways from how current implementations store resources. The deployment process involves configuring the LDP and loading the LDP dataset into it. It can be done in two ways based on the nature of the LDP server. First, if the LDP server accepts `POST` requests, an LDP Dataset Deployer can generate and send such requests for each resource contained in the LDP dataset. Second, using an LDP server that can directly consume the LDP dataset and expose resources from it. For now, our approach only requires the design document from which the entire LDP can be automatically generated.

3 Implementation

In this section, we describe our implementation which consists of a tool for every component from the LDP generation workflow.

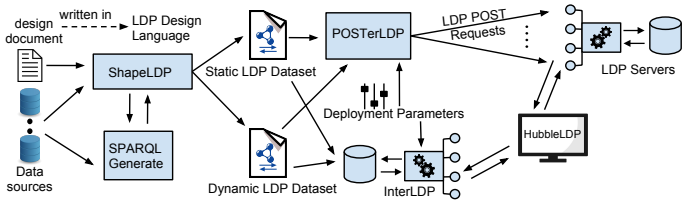

Fig. 2: Implementation of our LDP Generation Workflow

ShapeLDP [2] is an LDPizer that interprets documents written in LDP-DL refered to as *design documents*. To exploit heterogeneous data sources, lifting rules specified for `DataSource`s in SPARQL-Generate [6] are used. ShapeLDP can process design documents in *static evaluation* (resp. *dynamic evaluation*) to produce a *static* (resp. *dynamic*) *LDP dataset*. The difference between these two types of LDP dataset is that the static one contains the materialized RDF graph of the LDP resource while the dynamic one contains the information to generate this RDF graph and can thus be used for dynamic or real-time data sources. The algorithms used in ShapeLDP and the formal models of LDP dataset (static or dynamic) are given in our technical report [4]

InterLDP [3] is an LDP server which can directly consume an LDP dataset (static or dynamic) and expose resources from it. It was validated against the conformance tests of the LDP read interactions.[4]. In the static mode, it consumes static LDP dataset and exposes resources from it. In the dynamic mode, it is able to consider dynamic and real-time heterogeneous or RDF data sources and process the request for a particular resource and generate its RDF graph at query time.

POSTerLDP [5] is the implementation of the LDP Datset Deployer. It consumes a static LDP dataset and deployment parameters: base URL of LDP server and optionally the username and password for basic authentication on the server. It can funtion in two different modes: *append* and *update*. In the *append mode*, it sends only `POST` request to the server to create resources from the LDP dataset. In the *update mode*, for resources on the LDP having similar URLs with that from the LDP dataset, a `PUT` request with the RDF graph from the LDP dataset is sent for those resources.

HubbleLDP [6] is an LDP browser that can be used to browse resources on an LDP and view their content. Fig. 3 shows a screenshot which is actually an instance of it is running at `http://bit.ly/2BGYl9X` loaded with an LDP[7] about DCAT catalogue[8] and organization of its datasets in different languages.

[2] `https://github.com/noorbakerally/ShapeLDP`
[3] `https://github.com/noorbakerally/InterLDP`
[4] The conformance report is available at `https://w3id.org/ldpdl/InterLDP/execution-report.html`
[5] `https://github.com/noorbakerally/POSTerLDP`
[6] `https://github.com/noorbakerally/HubbleLDP`
[7] `http://opensensingcity.emse.fr/ldpdfend/tourism62/d3/catalog`
[8] `https://tourisme62.opendatasoft.com/api/v2/catalog/exports/ttl`

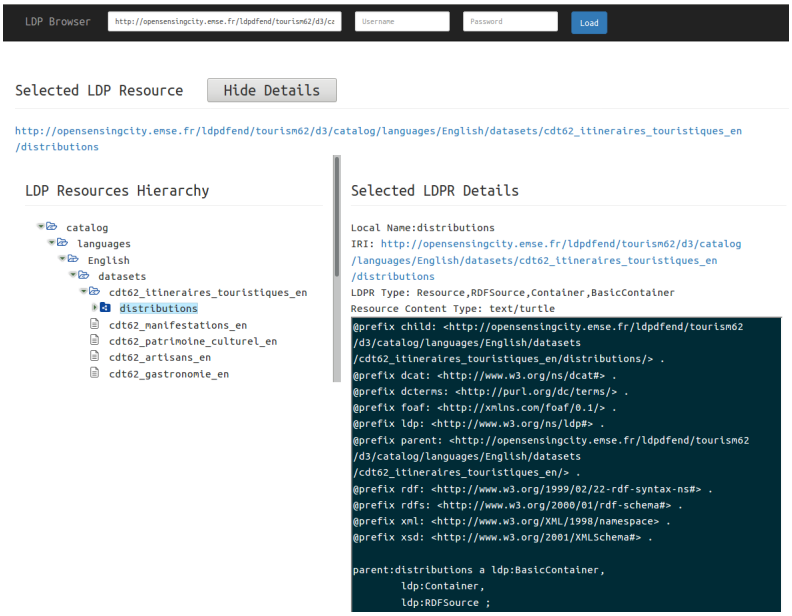

Fig. 3: Screenshot of HubbleLDP

4 Demonstration

We perform several experiments to show that our approach can significantly automatize the generation of LDPs from static, dynamic and heterogeneous data sources while keeping the design loosely coupled from the implementation. A description of all these experiments can be found on our GitHub page[9]. Below, we describes some of these experiments.

Reusability of design documents We perform two experiments to show that design documents are indeed reusable. In the first experiment[10], we consider 22 DCAT datasets from data portals as input data sources along 5 design documents. Every design document is applied of all data sources showing their reusability and on every data source, 5 design documents are applied showing the flexibility. In all, 110 LDPs are generated. In the second experiment[11], we consider two generic design documents that can we reused on RDF graph that uses RDFS/OWL vocabularies and apply them on one input data source to generate two LDPs with different designs. These two experiments shows that the design is highly reusable and not tighly coupled to any specific implementation.

Heterogeneous data sources [12] Handling heterogeneity of data portals is demonstrated by doing deploying 2 datasets,in JSON and CSV formats, via an LDP. In the design document, the original data source is specified together with a lifting rule. Using

[9] https://github.com/noorbakerally/LDPDatasetExamples
[10] https://tinyurl.com/y9rhgs6g
[11] https://tinyurl.com/yaoyt6kt
[12] https://tinyurl.com/yd8auvp2

SPARQL-Generate, the RDF data is generated and used by ShapeLDP to generate the LDP dataset which is finally deployed as an LDP using InterLDP.

Dynamic data sources [13] Then, we use a dataset which is being updated on a real-time basis and deploy it via an LDP to show that our approach can cope with hosting constraints. Using dynamic evaluation in ShapeLDP, the dynamic LDP dataset is generated and used by InterLDP to expose the LDP. Generating response for LDP-RSs takes more time because their content are generated at the query time using real-time data from the source.

Compatibility with existing LDP implementations [14] To show that our approach is compatible with existing LDP servers, we use POSTerLDP to deploy 2 LDPs over LDP servers that are instances of Apache Marmotta and Gold, both of them being reference implementation of the LDP standard.

5 Conclusion and Future Work

Linked Data Platforms can potentially ease the work of data consumers, but there is not much support from implementations to automate the generation and deployment of LDPs. Considering this, we proposed an approach, the LDP generation workflow, whose core is a language, LDP-DL, to generate LDPs. We describe the approach, give an implementation of it and demonstrates its ability to generate LDPs from static, dynamic, heterogeneous data sources while keeping the design loosely coupled from the implementation. Several improvements can be envisaged such as considering new design aspects (access rights, pagination etc.), support non-RDF sources and other types of LDP containers which we intend to consider in future versions.

Acknowledgments This work is supported by grant ANR-14-CE24-0029 from *Agence Nationale de la Recherche* for project OpenSensingCity.

References

1. N. Bakerally. Towards automatic deployment of linked data platforms. In *ISWC Doctoral Consortium*, 2017.
2. N. Bakerally. LDP-DL: RDF Syntax and Mapping to Abstract Syntax. Technical report, Mines Saint-Étienne, 2018. `https://w3id.org/ldpdl`.
3. N. Bakerally and A. Zimmermann. A system to automatize the deployment of data in linked data platforms. In *ISWC 2017 Posters & Demo*, 2017.
4. N. Bakerally, A. Zimmermann, and O. Boissier. LDP-DL: A language to define the design of Linked Data Platforms. Technical report, Mines Saint-Étienne, 2018. `http://w3id.org/ldpdl/technical_report.pdf`.
5. R. B. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *FOSE*, 2007.
6. M. Lefrançois, A. Zimmermann, and N. Bakerally. A SPARQL extension for generating RDF from heterogeneous formats. In *ESWC*, 2017.
7. S. Speicher, J. Arwe, and A. Malhotra. Linked Data Platform 1.0, W3C Recommendation 26 February 2015. Technical report, W3C, 2015.
8. T. Stahl, M. Volter, J. Bettin, A. Haase, and S. Helsen. *Model-driven software development: technology, engineering, management.* Pitman, 2006.

[13] `https://tinyurl.com/yb6cl3r7`
[14] `https://tinyurl.com/yc9yg8nz`