Towards automatic deployment of Linked Data Platforms

Noorani Bakerally

Univ Lyon, IMT Mines Saint-Étienne, CNRS, Laboratoire Hubert Curien UMR 5516, F-42023 Saint-Étienne, France
noorani.bakerally@emse.fr

1 Problem Statement

In the open data context, multiple data sources found at different locations on the Web generate massive heterogeneous sets of data. Using data from these data sources can provide useful information for decision making purposes in different domains. However, the levels of heterogeneity (e.g. syntactic, semantic, access etc.) render the exploitation of these data sources complex. Linked data platforms complying with the Linked Data Platform (LDP) standard [13], which we refer to as LDPs, can be used to facilitate data exploitation from these sources by providing a homogeneous view and access to one or more of them. However, the main problem is that putting in place an LDP in itself is complex. It can be decomposed into sub-problems categorized in the three main phases of an LDP development life cycle: design, implementation and deployment.

The design phase involves characterizing design aspects (e.g. characterizing content of Linked Data resources) and taking design decisions based on design choices (e.g. providing only triples where the Linked Data resource is the subject). Design decisions can be based on intuition and made without a proper justification or can also be supported by design guides. We refer to design guides as the set of standards, best practices, principles or design patterns which can be used to guide a design decision. The problem **P1** is that this phase is complex as design decision making process is manual and thus time consuming. Also knowledge of design guides is required to make proper design decisions.

After the design phase, the implementation phase starts the design decisions are encoded in the final product. The problem **P2** in this phase is that existing LDP solutions do not provide native support for decoupling the design and the implementation. As a result, using the solutions as they are can lead to tight coupling between the design and the implementation, hence, making it complex to maintain and reuse the design.

Finally, the implementation is deployed in a software environment. During the deployment phase of data-driven systems like LDPs, both system and data must be deployed. Considering our context of open data, there are data having hosting constraints, such as storage limitations or license restrictions, which arise when exploiting external constrained data sources. The problem **P3** in this phase is that the constraints prevent from deploying a copy of the data in the software environment thus making it complex to directly exploit these constrained data sources.

2 Relevancy

Having stated the problems (**P1**, **P2**, **P3**) that we consider, lets turn now to motivate and stress their relevance using a concrete case of smart cities within the open data context which we are dealing with in the OpenSensingCity[1] project.

The governmental body responsible for a city may want to facilitate access to city data so that it may be utilized for useful ends such as developing smart city applications which can aid citizens in their daily activities. In this case, an LDP may be used as it can uniformize access to data thus enabling much interoperability. A city is a decentralized ecosystem having different organizations which can provide data. As such, the city LDP may itself have to access and exploit other data sources which may be owned and controlled by other organizations operating in different domains (e.g transportation, parking). There may be organizations wishing to participate in this effort by opening their data via LDPs. However, they may be reluctant as they may not want to invest much time in developing LDPs (cf. **P1**). Therefore support, such as automated solutions, which conforms to standards and best practices, may help to speed the design. Besides this gain of time, it may help engineers who may not master current standards and best practices, to guarantee the respect of these standards and best practices in the publication of their data.

Considering the above case, there may be new organizations opening their data and thus require maintaining the design of the city LDP to consider the new data sources. A tight coupling between the design and implementation (cf. **P2**) may make the maintenance complex and lengthen this process. Moreover, there may be another city which may also want to expose data using an LDP having a similar design. Again, due to the tight coupling issue (cf. **P2**), it may not be possible to directly reuse the design and thus requiring the other city to re-encode the design in the implementation. If the design is completely decoupled from the implementation, it becomes highly reusable and can be directly applied to another LDP. Applying this in the city scenario, different cities may reuse a design, exposing data in the same way. As a result, generic smart city applications may be developed for different purposes, such as finding parkings or transportation modalities. These applications may exploit any city LDP as long as the LDPs use a design and vocabulary known by the application.

If hosting constraints (cf. **P3**) are not taken into consideration, manual development of data adapters may be required to exploit constrained data sources. Considering the above case, city LDPs may have to exploit data sources from organization which are bounded by licenses or data from sources like LinkGeoData[2] which may be difficult to host due to storage limitations. Providing support for hosting constraints may facilitate exploitation of these constrained data sources without having to perform further manual development.

[1] `http://opensensingcity.emse.fr`
[2] `http://linkedgeodata.org` on 1 May 2017

3 Related Work

There are solutions for publishing data via a linked data platform. Among these solutions, some conform to the LDP standard and others do not. Non-LDP conformant solutions are important to consider because even if they do not comply with the LDP standard, they minimally address some problems that we consider. Pubby[3], D2R Server [3], Virtuoso [4], Triplify [1]) can automatically deploy linked data from existing RDF data sources or data sources having a virtual RDF representation. Triplify and D2R, have been designed to expose relational data as linked data. As such, they are more focused on defining mappings between relational database schemas and RDF vocabularies. The final step of providing the RDF as linked data mostly only involves ensuring that RDF resources can be dereferenced with RDF data. In addition to providing relational data as linked data, Virtuoso also provides a linked data interface to its triple store. Pubby can provide a linked data interface both to SPARQL endpoints and static RDF documents. Pubby and Virtuoso are the only tools for directly publishing RDF data as linked data. However, most design decisions are fixed and cannot be parameterized. For example, it is not possible to specify the content to provide for RDF resources. In addition, they rely on implementation-specific details. For example, to obtain content for RDF resources, Pubby uses SPARQL DESCRIBE queries which are implementation specific and determined by the SPARQL query processor. In summary, related to our problems described in Section 1, non-LDP conformant solutions address **P1** by providing some level of automization but the automatized design decisions seem to be based on intuition. **P2** is considered by using a vocabulary to declaratively describe the design separately from the implementation. However, the vocabulary has a low expressivity resulting in design aspects which cannot be characterized such as specifying the content definition of an linked data resources. **P3** is partly addressed as the solutions can be used to deploy linked data platform on some constrained data sources. However, there are some issues which are left aside such as the treatment of blank nodes, external resources and entailment regimes.

Linked Data Platform 1.0 [13] is a W3C recommendation that provides a set of rules for read-write linked data via HTTP. As of now, we refer to the *LDP standard* as the W3C recommendation and *LDP* as a data platform conforming to this standard. Data resources exposed via LDPs are referred as LDP Resources (LDPR). LDPRs can be organized in collections, referred to as LDP containers. LDP containers are specialization of LDPR which helps organizing other resources (e.g LDP container members). There are LDP solutions. We restrict our analysis of them to those mentioned in LDP implementation conformance report[5] which show their degree of conformance to the LDP standard. We categorize solutions implementing the LDP standard into LDPR management system (Cal-

[3] `http://wifo5-03.informatik.uni-mannheim.de/pubby/` on 18 May 2017
[4] `https://virtuoso.openlinksw.com` on 19 July 2017
[5] `https://www.w3.org/2012/ldp/hg/tests/reports/ldp.html` on 19 July 2017

limachus[6], Carbon LDP[7], Fedora Commons[8], Apache Marmotta[9], Virtuoso[10], gold[11], rww-play[12]) and LDP framework (Eclipse Lyo[13], LDP4j [7, 8]). LDPR management system can be seen as a repository for LDPRs on top of which CRUD operations, adhering to the LDP standard, are allowed through HTTP methods. LDP frameworks are solutions which can be used to build custom applications which implement LDP interactions. LDP-conformant solutions are in their early stages as there is no support for deploying even existing RDF data via an LDP. To do so, user has to take all the design decisions related to linked data publication such as IRI definitions for RDF Web documents, structure of these document and their content definition and write custom data adapters for generating LDPRs to materialize them in an LDP. Based on our analysis, we believe these solutions solutions suffer from all the problems mentioned in Sec. 1.

4 Research Questions and Hypotheses

From the problems and analysis of the state of the art, the main research question of our work is whether it is possible to automatically deploy an LDP. This question in turn generates many other questions which are centered around two main aspects, design (**RQ1**, **RQ2**) and deployment (**RQ3**, **RQ4**) of LDPs. The research questions and their hypotheses are:

RQ1 How to decouple the design from the implementation of LDPs?
- We hypothesize that the design of LDPs can be described declaratively in a separate document using a dedicated vocabulary.

RQ2 How to generate the design data sources?
- We hypothesize that a default design document for an LDP can be generated using the ontologies associated with the data sources.

RQ3 How to automatize deployment of an LDP?
- We hypothesize that the deployment of an LDP can be automatized using its design document.

RQ4 How to consider hosting constraints when deploying LDP?
- We hypothesize that hosting constraints can be considered by generating LDP resources on the fly using the design document on the LDP.

5 Proposed Approach

To answer our research questions, we provide an approach which uses principles from model-driven engineering. MDE provides general principles for addressing the problems mentioned in Sec. 1. In summary, it involves using models

[6] `http://callimachusproject.org` on 15 July 2017
[7] `http://carbonldp.com` on 15 July 2017
[8] `http://fedora-commons.org` on 15 July 2017
[9] `http://marmotta.apache.org` on 15 July 2017
[10] `http://www.openlinksw.com/` on 15 July 2017
[11] `https://github.com/linkeddata/gold` on 15 July 2017
[12] `https://github.com/read-write-web/rww-play` on 15 July 2017
[13] `http://wiki.eclipse.org/Lyo/LDPImpl` on 15 July 2017

as first-class entities and transforming them into running systems by using generators or by dynamically executing the models at run-time [9,10]. By separating models from the running systems, MDE enables separation of concerns thus guaranteeing higher maintainability of software systems and reusability of systems' models [14]. Therefore, in our approach, we intend to apply MDE principles in the context of LDPs to automatize ther design and deployment.

Figure 1 shows a general overview of our approach. The circled numbers in the text below refer to the corresponding circles from the figure. Each of the circled numbers are components. They are described below and relate to research questions in Sec. 4. (1) relates to **RQ1**, (2) relates to **RQ2**, (3) and (4) relates to **RQ3** and finally (3) and (5) relates to **RQ4**.

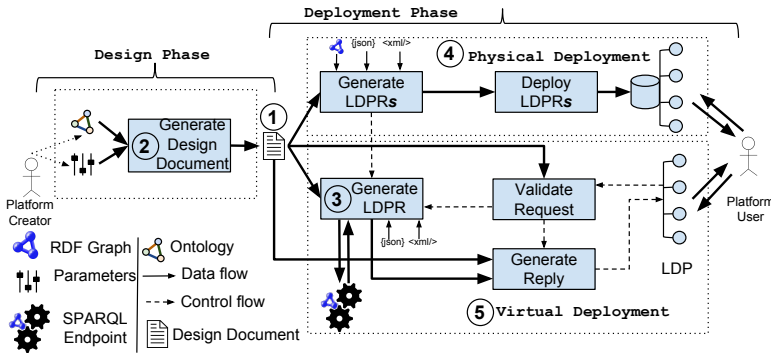

Fig. 1. General Overview of our Approach

5.1 Design Phase Support

The aim of the design phase is to produce the design document (1). To decouple the design from the implementation of an LDP, we will provide a vocabulary to declaratively describe the design in the design document separately from the implementation. To further facilitate the creation of this document, we will provide a method (2) to automatically produce it from existing data sources. In the next two paragraphs, we describe the vocabulary and (2).

The vocabulary will be formalized using ontology language, such as RDFS or OWL, based on the required level of expressivity. The terms from the vocabulary will correspond to design aspects and their respective design choices. To set up this vocabulary, first, we will identify the design aspects of an LDP. Design aspects can be identified from existing linked data platforms and also from standards and best practices. For instance, the following elements issued from the LDP standard are good starting points: organization of LDP resources in terms of LDP containers, LDP container types, content definition of LDP containers and their members and IRI structure of LDP container and their members. After design aspects have been identified, a set of design choices for each of design

aspect has to be established. For some, established design choices already exist. For others, no established design patterns exist(e.g. to characterize the content of an LDP container or their members). To this aim, we intend to perform experiments on current deployed Linked Data to analyze current trends, derive design patterns and set up design choices based on them.

The core aspect of (2) is to generate the design document by automating generic design decisions. An example of a generic decision can be to use the class hierarchy of the ontology to generate LDP containers. All generic design decisions made will be based on design patterns which will be derived from our experiments or analysis. We intend to make (2) parameterizable to provide some level of customizations and allow the user to have some control over the generated design document. (2) will generate the design document using the metamodel of the data and possibly some parameters provided by the platform creator. Related to the metamodel, we consider two types of data, RDF data and Non-RDF data (CSV, JSON, XML, etc.). For RDF data, the metamodel can either be an explicit ontology or the ontology implicit in the data. For Non-RDF data (CSV, JSON, XML, etc.), the metamodel is the ontology used for RDFizing the data. Although we do not explicitly address data heterogeneity, we will use existing works in this area such as *GRDDL* [4], *R2RML* [5], *RML* [6] and *SPARQL Generate* [11].

5.2 Deployment Phase Support

We consider two strategies for supporting deployment of LDPs, **`Physical Deployment`** (4) and **`Virtual Deployment`** (5). The process *Generate LDPR* (3) is based on a method, which we will introduce, and is common in both strategies. (3) will use the design document, extract the appropriate design decisions made for LDPRs and generate the content for them. In (4), (3) is used to generate all the LDPRs and materialize them in one go in an LDP. In (5), on obtaining a request for an LDPR, there is a validation process after which the LDPR is generated using (3). In contrast to (4), in (5), the LDPRs cannot be materialized. Instead, data resources remain in their external data sources in their native state. Serving them as LDPRs requires querying these data sources, applying the required transformation to generate them.

As such, one potential problem with (5) is the extra time taken to generate the reply compared to (4). One reason why this may occur is due to having a dynamic controller which exploits the design document every time a request is obtained to validate and process it. A possible solution may be to have different types of controllers namely:

- Semi-dynamic controllers: Dedicated controllers are generated for different categories of LDPRs. The required segments from the design document is encoded in the generated dedicated controllers to avoid considering the design document for processing every request. The controllers may be re-generated if the design document changes.
- Dynamic controllers: Only one generic controller is used and the design document is considered when processing every request.

Components (3) and (4) are our two main strategies for supporting deployment. We also intend to consider hybrid deployment strategies using both (4) and (5) such as where some LDPRs are materialized in data stores while some others need to be generated on the fly at query time.

6 Evaluation Plan

We intend to make the evaluation more robust as the research matures and also using feedback from this doctoral consortium. As of now, our evaluation is structured as shown below:

1. **Flexibility**: Show the flexibility of our approach by performing experiments in different concrete scenarios using real datasets. These experiments will also answer our research quetions. We intend to:
   (a) use similar design for LDPs having different data sources and use different designs for LDPs with similar data sources (cf. **RQ1**);
   (b) use linked data to build a design document where modular parts of other design document are combined to obtain the final design document (cf. **RQ1**);
   (c) directly use generated design documents from RDF datasets for automatically deploying them as LDPs (cf. **RQ2**, **RQ3**);
   (d) deploy LDP on top of dynamic smart city data sources having hosting constraints (cf. **RQ4**).
2. **Quality**: Evaluate the automatized design with respect to the quality of Linked Data it generates using an evaluation framework which we will introduce based on *Data on the Web Best Practices* [12]. For best practices from [12], we will setup metrics wherever possible as some best practices are purely qualitative;
3. **Performance**: Evaluate performance of Physical Deployment against Virtual Deployment. This will also involve evaluating semi-dynamic controllers against dynamic controllers in Virtual Deployment to provide an insights of the optimization which is possible using semi-dynamic controllers;

Finally, since we are using an MDE approach, we intend to investigate the state of the art for the evaluation of such approaches and apply them, wherever possible, in our approach.

7 Reflections

Some of the works performed in our PhD have been good starting points for understanding the context of work discussed in this paper. We developed from scratch SCANS [2] which is a linked data platform for smart city artifacts. Doing so has broaden our knowledge of design and deployment of linked data platforms and enabled us to see the different problems which can arise when developing them. We experimented with data platform solutions and found that there are tasks which are repetitive and can be automated. We also found that vocabularies are already in use (cf. Sec.3) to describe some minimal design aspects which shows that it is possible to generate a data platform using a design

description. While writing custom data adapters to generate data resources, we found that they can be generalized and changing parameters be externalized, hence making room to consider hosting constraints. We have also seen patterns both in the IRIs and content of linked data resources. For example, we have noticed that the content of linked data resources includes those triples from the original RDF graph where the resource is either the subject or the object. This pattern can be formalized as a design choice. More such patterns exists in current deployed Linked Data and we will perform experiment to find relevant ones for our works. Moreover, after going through the different best practices provided by [12], we believe that our approach will be able to conform to many of them thus guaranteeing publishing data conforming to best practices. Based on concrete observations mentioned, we believe that our approach can be successful.

Acknowledgments This work is supported by ANR grant 14-CE24-0029 for project OpenSensingCity. The author would also like to acknowledge the thesis's supervisors, Antoine Zimmermann and Olivier Boissier.

References

1. S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller. Triplify: light-weight linked data publication from relational databases. In *Proceedings of the 18th international conference WWW*. ACM, 2009.
2. Noorani Bakerally, Olivier Boissier, and Antoine Zimmermann. Smart city artifacts web portal. In *ESWC*. Springer, 2016.
3. Christian Bizer and Richard Cyganiak. D2R server-publishing relational databases on the semantic web. In *Poster at the 5th ISWC*, volume 175, 2006.
4. Dan Connolly et al. Gleaning resource descriptions from dialects of languages (GRDDL). *W3C, Recommendation REC-grddl-20070911*, 2007.
5. Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language, W3C Recommendation 27 September 2012. Technical report.
6. A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and Rik Van de Walle. RML: A generic language for integrated RDF mappings of heterogeneous data. 2014.
7. M. Esteban-Gutiérrez, N. Mihindukulasooriya, and R. García-Castro. Interoperable read-write linked data application development with the LDP4j framework.
8. M. Esteban-Gutiérrez, N. Mihindukulasooriya, and R. García-Castro. LDP4j: A framework for the development of interoperable read-write linked data applications. 2014.
9. Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *2007 FSE*. IEEE Computer Society, 2007.
10. Jochen Küster. Model-driven software engineering foundations of model-driven software engineering. *IBM Research*, 2011.
11. M. Lefrançois, A. Zimmermann, and N. Bakerally. A SPARQL extension for generating rdf from heterogeneous formats. In *14th ESWC 2017*, 2017.
12. Bernadette Farias Lóscio, Caroline Burle, and Newton Calegari. Data on the web best practices. Working draft, W3C, January 2016.
13. Steve Speicher, John Arwe, and Ashok Malhotra. Linked Data Platform 1.0. Technical report, World Wide Web Consortium (W3C), February 26 2015.
14. Thomas Stahl and Markus Volter. *Model-driven software development: technology, engineering, management*. J. Wiley & Sons, 2006.