Generation of Linked Data Platform in Highly Decentralized Information Ecosystem

Mohammad Noorani Bakerally

N°d'ordre NNT : 2018LYSEM029

THESE de DOCTORAT DE L'UNIVERSITE DE LYON
opérée au sein de
l'Ecole des Mines de Saint-Etienne

Ecole Doctorale N° 488
Sciences, Ingénierie, Santé

Spécialité de doctorat :
Discipline : Informatique

Soutenue publiquement/à huis clos le 20/12/2018, par:
Mohammad Noorani BAKERALLY

Génération de plateforme de données liées dans un écosystème d'informations fortement décentralisé

Devant le jury composé de :

Présidente:LAFOREST Frederique, Professeur, Université Jean Monnet
Rapporteur: SKAF-MOLLI Hala, Maître de conférences, Université de Nantes
Rapporteur: CHAMPIN Pierre-Antoine, Maître de conférences, Univ. Claude Bernard Lyon 1
Examinatrice: VIDAL Maria-Esther, Professeur, Universidad Simón Bolívar
Invité: NOULARD Eric, Ingénieur de recherche, Antidot
Directeur de thèse: BOISSIER Olivier, Professeur, École des Mines de Saint-Étienne
Co-directeur de thèse: ZIMMERMANN Antoine, Maître Assistant, École des Mines de Saint-Étienne

Spécialités doctorales	Responsables :	Spécialités doctorales	Responsables
SCIENCES ET GENIE DES MATERIAUX	K. Wolski Directeur de recherche	INFORMATIQUE	O. Boissier, Professeur
MECANIQUE ET INGENIERIE	S. Drapier, professeur	SCIENCES DES IMAGES ET DES FORMES	JC. Pinoli, Professeur
GENIE DES PROCEDES	F. Gruy, Maître de recherche	GENIE INDUSTRIEL	N. Absi, Maitre de recherche
SCIENCES DE LA TERRE	B. Guy, Directeur de recherche	MICROELECTRONIQUE	Ph. Lalevée, Professeur
SCIENCES ET GENIE DE L'ENVIRONNEMENT	D. Graillot, Directeur de recherche		

EMSE : Enseignants-chercheurs et chercheurs autorisés à diriger des thèses de doctorat (titulaires d'un doctorat d'Etat ou d'une HDR)

ABSI	Nabil	MR	Génie industriel	CMP
AUGUSTO	Vincent	MR	Génie industriel	CIS
AVRIL	Stéphane	PR	Mécanique et ingénierie	CIS
BADEL	Pierre	PR	Mécanique et ingénierie	CIS
BALBO	Flavien	PR	Informatique	FAYOL
BASSEREAU	Jean-François	PR	Sciences et génie des matériaux	SMS
BATTON-HUBERT	Mireille	PR	Sciences et génie de l'environnement	FAYOL
BEIGBEDER	Michel	MA	Informatique	FAYOL
BILAL	Blayac	DR	Sciences et génie de l'environnement	SPIN
BLAYAC	Sylvain	PR	Microélectronique	CMP
BOISSIER	Olivier	PR	Informatique	FAYOL
BONNEFOY	Olivier	PR	Génie des Procédés	SPIN
BORBELY	Andras	DR	Sciences et génie des matériaux	SMS
BOUCHER	Xavier	PR	Génie Industriel	FAYOL
BRUCHON	Julien	PR	Mécanique et ingénierie	SMS
CAMEIRAO	Ana	PR	Génie des Procédés	SPIN
CHRISTIEN	Frédéric	PR	Science et génie des matériaux	SMS
DAUZERE-PERES	Stéphane	PR	Génie Industriel	CMP
DEBAYLE	Johan	MR	Sciences des Images et des Formes	SPIN
DEGEORGE	Jean-Michel	MA	Génie industriel	Fayol
DELAFOSSE	David	PR	Sciences et génie des matériaux	SMS
DELORME	Xavier	PR	Génie industriel	FAYOL
DESRAYAUD	Christophe	PR	Mécanique et ingénierie	SMS
DJENIZIAN	Thierry	PR	Science et génie des matériaux	CMP
BERGER-DOUCE	Sandrine	PR	Sciences de gestion	FAYOL
DRAPIER	Sylvain	PR	Mécanique et ingénierie	SMS
DUTERTRE	Jean-Max	PR	Microélectronique	CMP
EL MRABET	Nadia	MA	Microélectronique	CMP
FAUCHEU	Jenny	MA	Sciences et génie des matériaux	SMS
FAVERGEON	Loïc	MR	Génie des Procédés	SPIN
FEILLET	Dominique	PR	Génie Industriel	CMP
FOREST	Valérie	PR	Génie des Procédés	CIS
FRACZKIEWICZ	Anna	DR	Sciences et génie des matériaux	SMS
GAVET	Yann	MA	Sciences des Images et des Formes	SPIN
GERINGER	Jean	MA	Sciences et génie des matériaux	CIS
GONDRAN	Natacha	MA	Sciences et génie de l'environnement	FAYOL
GONZALEZ FELIU	Jesus	MA	Sciences économiques	FAYOL
GRAILLOT	Didier	DR	Sciences et génie de l'environnement	SPIN
GROSSEAU	Philippe	DR	Génie des Procédés	SPIN
GRUY	Frédéric	PR	Génie des Procédés	SPIN
HAN	Woo-Suck	MR	Mécanique et ingénierie	SMS
HERRI	Jean Michel	PR	Génie des Procédés	SPIN
ISMAILOVA	Esma	MC	Microélectronique	CMP
KERMOUCHE	Guillaume	PR	Mécanique et Ingénierie	SMS
KLOCKER	Helmut	DR	Sciences et génie des matériaux	SMS
LAFOREST	Valérie	DR	Sciences et génie de l'environnement	FAYOL
LERICHE	Rodolphe	DR	Mécanique et ingénierie	FAYOL
LIOTIER	Pierre-Jacques	MA	Mécanique et ingénierie	SMS
MOLIMARD	Jérôme	PR	Mécanique et ingénierie	CIS
MOULIN	Nicolas	MA	Mécanique et ingénierie	SMS
MOUTTE	Jacques	MR	Génie des Procédés	SPIN
NAVARRO	Laurent	MR	Mécanique et ingénierie	CIS
NEUBERT	Gilles	PR	Génie industriel	FAYOL
NIKOLOVSKI	Jean-Pierre	Ingénieur de recherche	Mécanique et ingénierie	CMP
O CONNOR	Rodney Philip	PR	Microélectronique	CMP
PICARD	Gauthier	PR	Informatique	FAYOL
PINOLI	Jean Charles	PR	Sciences des Images et des Formes	SPIN
POURCHEZ	Jérémy	DR	Génie des Procédés	CIS
ROUSSY	Agnès	MA	Microélectronique	CMP
SANAUR	Sébastien	MA	Microélectronique	CMP
SERRIS	Eric	IRD	Génie des Procédés	FAYOL
STOLARZ	Jacques	CR	Sciences et génie des matériaux	SMS
VALDIVIESO	François	PR	Sciences et génie des matériaux	SMS
VIRICELLE	Jean Paul	DR	Génie des Procédés	SPIN
WOLSKI	Krzystof	DR	Sciences et génie des matériaux	SMS
XIE	Xiaolan	PR	Génie industriel	CIS
YUGMA	Gallian	MR	Génie industriel	CMP

Mise à jour : 24/11/2020

École Nationale Supérieure Des Mines De Saint-Étienne

Abstract

Doctor of Philosophy

Generation of Linked Data Platforms from existing data sources in highly decentralized information ecosystems

by Mohammad Noorani Bakerally

Information ecosystem with decentralized architectures have known some success in terms of data interoperability. However, today, information ecosystems in which organizations are operating are moving towards highly decentralized architectures for reasons such as globalization, multinationalism or complex supply chain collaborations. The problem in highly decentralized information ecosystem is that while data is heterogeneous, there is little to no coordination between its different units. Consequently, resolving data heterogeneity and enhancing interoperability between the information systems become more challenging in such a context.

In this work, our hypothesis is that Semantic Web technologies and in particular the Linked Data Platform 1.0 (LDP) standard can be used to provide a homogeneous view and access to self-described data. Thus, it decreases the need for contacting data providers to understand and make use of the data. However, while several LDP implementations exist, they provide no support for automating the generation of LDPs. Consequently, deploying LDPs from existing data sources still involves a lot of manual development. To solve this problem, we propose an approach that uses a language to describe how existing data sources can be used to generate LDPs compatible with any implementation of the LDP standard and deployable on any of them. We formally describe the syntax and semantics of language. We provide an implementation of the approach that instantiates an automatized generation and deployment workflow. Finally, we evaluate our language and approach in general by performing several experiments to show how our approach automatizes the generation of LDPs, while enhancing design reusability, from existing data sources that are either heterogeneous or have hosting constraints.

Acknowledgements

Doing this PhD and writing this thesis would not have been possible without the help of many people whom I met in different spheres of my life.

I wish to express my deepest gratitude to my supervisors, Antoine Zimmermann and Olivier Boissier. I thank them for having seen in me the ability to embark on this journey. Their knowledge and experience have been of great help in the writing of this thesis. I am also grateful to them for having dedicated their time to guiding me and providing me with their precious insights and advice. Last but not least, I admire their passion and humility during these three years which made it a pleasure to work with them.

I thank the members of the jury for accepting to evaluate this thesis, namely: Frederique Laforest for presiding the comity of jury, Hala Skaf-Molli and Pierre-Antoine Champin for reviewing this work, and Maria-Esther Vidal and Eric Noulard for examining this work. I also thank them for their constructive comments and questions that have lead to a fruitful discussion on this work.

I thank Maxime Lefrançois who gave me the opportunity to collaborate in SPARQL Generate. Doing so has allowed me to think about my thesis. Also, I thank my friend, Khadim Ndiaye, for the numerous times he helped me in difficult times during my stay in Saint-Etienne. I shall always cherish the interesting discussions we had on varying issues and the nice dishes we made together on Friday nights.

Down memory lane, some people believed in me right from the start and motivated me to continue my academic pursuits. I am indeed thankful to them. Anwar Chutoo, my supervisor and lecturer at undergraduate level, encouraged me to gain expertise in this field right after completing my bachelor degree when I wanted to stop. Sharing with me his experience of learning computer science in a french system, Avinash Meetoo, my colleague, inspired me to do a PhD in France. While working in his company, Knowledge Seven, I also acquired many skills that better equipped me for this PhD. I also thank Assma Benharkat and Pierre-Nicolas Tollite for sharing useful information with me during our study at the University of Manchester, hence further fuelling the idea of a PhD in France.

Finally, I thank my parents whose upbringing made me reach this point in my life where I am actually writing this sentence. Special thanks to my sisters for all their efforts in facilitating this journey. At last, I thank my wife and best friend, Lindia, who came into my life in the last year of the PhD. The love and happiness she brought along made it easier to complete this work.

Contents

List of Figures

List of Tables

Introduction

Information ecosystems within organizations started with centralized architectures where a single information system would support all processes. However, maintaining such information systems was complex and they often ended as big systems with complex structures. Decentralized information ecosystems emerged as a solution with smaller information systems delineated towards specific processes. These information systems are managed by units, taking the role of data providers or consumers, that coordinate among themselves enhance interoperability between their information systems by resolving the required data heterogeneity [Hug07].

Today, for reasons such as globalization, multinationalism or complex supply chains, there is a special type of information ecosystem that is emerging from current decentralized ones that we refer to as *highly decentralized information ecosystem*. We call it 'highly decentralized' due to the self-governance of its units. This self-governance gives rise to the *information problem* that happens when information systems "erect barriers between themselves thereby reducing information flow" [Fox96, Gia04]. Consequently, there is a lack of coordination between these units making it difficult to for them to enhance interoperability bwtween their information systems. A concrete example of a highly decentralized information ecosystem is the open data context whose abstract view is shown in Figure 1.

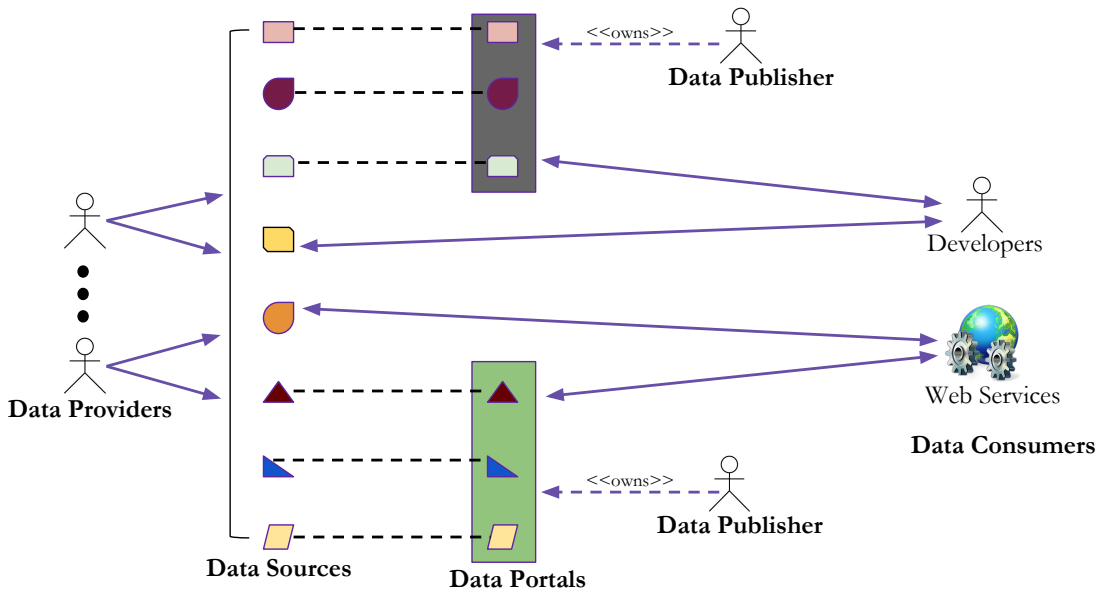

FIGURE 1: Overview of Open Data Context

In short, in this context, there are open data sources exposed by data providers that may be exploited by data consumers. The barriers between data providers and consumers are evident as there is no explicit contract between them, thus making

the information problem a fact. Consequently, data consumers may be unaware of data sources exposed by data providers. For this reason, there are data publishers that curate data sources from data providers and provide access to them via data portals. This is why data portals have become the main entry point to access data sources in the open data context.

While open data portals to some extent resolve the information problem and facilitate the discovery of data sources, data exploitation remains complex for different reasons. Firstly, after using data portals to discover data sources, it may be complex for data consumers to exploit them if the systems exposing them are heterogeneous or if they do not have the required metadata. Secondly, even if they have the required metadata, it may be heterogeneous as different data providers or publishers may use different standards and technologies for encoding it. Finally, data consumers may request the intervention of data providers for enhancing interoperability between their information systems. However, data providers may not intervene because as mentioned before, they is no explicit contract between them.

The open data context is only an instance of a highly decentralized information ecosystems. Other instances presenting similar problems may be found in inter and intra organizational information ecosystems.

Aim & Hypotheses

Given the problems in highly decentralized information ecosystems, our aim is to facilitate data exploitation by providing data consumers with a homogeneous view and access to the data and its semantics. To achieve this aim, Semantic Web standards and technologies that were originally conceived as an extension for the Web can be of great use by enabling the creation of information ecosystems where data can be exposed with well-defined semantics using standard data access mechanisms [Wu01, SAD+15]. Also, Semantic Web standards are known to be compatible for decentralized data management [SS06, Ska17].

To provide a standard access to data and their semantics, Semantic Web uses several open standards mainly RDF, RDFS/OWL, SPARQL and Linked Data Platform. RDF is a flexible data model that can integrate data from several sources. RDFS and OWL are ontology languages that can be used to formalize data semantics. SPARQL enables the extraction of data from RDF stores and finally the Linked Data Platform standard enables providing a homogeneous view and access to RDF data.

To achieve our aim using Semantic Web standards, we make the following hypotheses:

- Semantic Web standards can be used to enhance data interoperability in highly decentralized information ecosystems;
- Linked data platforms generated from existing data sources and conforming to the Linked Data Platform standard can be used to provide a homogeneous view and access to the data.

Objective

Based on the above hypotheses, in this thesis, our objective is to generate LDPs from existing data sources with respect to the following constraints:

Heterogeneity In highly decentralized information ecosystems, data sources can be can be heterogeneous when they are exposed by independent units having little to no coordination them. More precisely, data from these sources can be in different syntaxes, use terms having varying semantics, have diverse structures and be available via different access mechanisms.

Hosting Constraints Besides heterogeneous data sources, in highly decentralized information ecosystem, there data sources whose exploitation may give rise to hosting constraints. They prevent from hosting a copy of the data in a third-party environment and can be on the data itself (e.g. license restrictions), or can be a limitation of the third-party software environment (e.g. bandwidth or storage limitations to continuously verify and maintain fresh copies of dynamic or real-time data). Therefore, LDP in highly decentralized information ecosystems has to be able to cope with hosting constraints.

Design Reusability LDPs are data-driven system and their designs influence the view of the data they provide. As mentioned above, LDPs can be used to provide a homogeneous view and access to data. Nevertheless, having LDPs that use different designs to expose data sources may enhance view heterogeneity. By making the design of LDPs reusable, different LDPs may share similar designs and thus expose data without causing view heterogeneity. Therefore, in highly decentralized information ecosystem, different LDPs should be able to share similar designs

Evaluation Criteria

The criteria for evaluating our work is related to our objective and the constraints described in the previous section. We evaluate our approach presented in this thesis by considering whether it is able to automatize the design and deployment of LDPs from existing data sources with respect to the following:

- Data Heterogeneity: existing data sources can be heterogeneous;
- Hosting constraints: existing data sources can have hosting constraints;
- Design reusability: it must be possible to generate several LDPs using the same design.

Contribution

The major contributions of this thesis are:

- LDP generation workflow: This workflow enables the deployment of LDPs from existing data sources that may be heterogeneous or on which there may be hosting constraints;

- LDP Design language: Using this language, the design of an LDP can be described independent of its implementation and deployment. Thus, the design itself can be reused for different LDPs. Also, the instantiation of an LDP from the design can be implemented and deployed in different ways.
- Implementation agnostic deployment: LDPs can be deployed on LDP servers without considering their implementation specific details.
- Dynamic LDP: LDP can be deployed without having to host their data as they can dynamically exploit data sources at runtime. In this way, they can deal with real-time data sources or data sources having hosting constraints.

Thesis Outline

This thesis consists of four main parts. The first two parts relates to our hypotheses and the third part describes our implementation and evaluation and the last part presents our conclusion together with limitation and future perspectives. We detail these four parts below.

Semantic Web standards for highly decentralized information ecosystem

In Part I, we answer our first hypothesis by providing a set of requirements for enhancing data interoperability in highly decentralized information ecosystem and showing that Semantic Web standards satisfy these requirements. More precisely, in Chapter 1, we discuss different types of information ecosystems and introduce highly decentralized information ecosystems. We discuss the barriers to data interoperability in this information ecosystem and identify a set of requirements for enhancing it.

Then, in Chapter 2, we describe standards from the Semantic Web that satisfy the latter requirements and also describe existing tools that facilitate their implementation and usage. We position LDPs as the final step to provide a homogeneous view and access to existing data sources in highly decentralized information ecosystem and define requirements for automating them. While describing LDP related works, we show that none of them satisfy these requirements.

Model-Driven LDP Generation

In Part II, we answer our second hypothesis by presenting a model-driven approach to automatize the generation of LDPs having as its core element a language, LDP-DL, to define the LDP designs. We start Chapter 3 with a general description of an LDP development life cycle with a focus on manual tasks. Then, we describe model-driven engineering principles and integrate them in the development life cycle to finally obtain a generalized LDP generation life cycle whose core are languages to describe LDP design and deployment aspects. Then, we describe a set of design and deployment aspects, highlight those that we consider. Finally, based on the generalized LDP generation life cycle and LDP design and deployment aspects that

we consider, we present the LDP generation workflow that automatizes the design and deployment of LDPs.

Core to the LDP generation workflow is LDP-DL, the subject of Chapter 4, that is our language for defining the design of LDPs. We provide an overview of the language and proceed with the formal description of its syntax and semantics. The semantics of an LDP-DL design is an LDP dataset that is an abstract model to store LDP resources on an LDP. We define the semantics of our language in two different ways, model-theoretic and operational. The model-theoretic semantics enable us to abstract from choices left open by the LDP standard and associate an LDP dataset to an LDP-DL design. The operational semantics instantiate the model-theoretic semantics to evaluate a LDP-DL design and generate an LDP dataset containing the LDP resources. Finally, we describe the variabilities that may invalidate LDP datasets and provide two intermediary models, static and dynamic data sources, to abstract them.

Implementation & Evaluation

Part III is focused on the implementation and evaluation of our proposal with respect to the evaluation criteria defined above. In Chapter 5, we describe the LDP generation toolkit that is an implementation of the workflow presented in Chapter 3. We start with a brief description of the concrete syntax of LDP-DL and describe how a design document in this syntax can be used to generate LDP datasets. Then, we describe the instantiation of LDPs from LDP datasets on servers while remaining agnostic of their implementation-specific details. We also describe our compatible LDP server that can also do so.

Finally, in Chapter 6, we perform several experiments to evaluate our work based on our evaluation criteria . We show that LDP-DL allows defining reusable LDP designs that can be either dependent or independent of the ontologies used at the data sources. Moreover, we generate LDPs from heterogeneous data sources to show that our approach can deal with data heterogeneity. We instantiate dynamic LDPs to demonstrate the ability to deal with hosting constraints. As for automated LDP generation, all LDPs generated in our experiments are automated. But to further reinforce this aspect, we automate deployment of LDPs on servers while remaining agnostic of their implementation-specific details. Besides, we also discuss side contributions of our approach namely flexibility and lightweight data integration.

Conclusion & Perspectives

Finally, in Part IV, we provide a summary of our work and highlights our contributions. Then, we describe its limitations as well as the avenues for future researches that it opens.

Part I

State of the Art

Chapter 1

The Data Arena in Highly Decentralized Information Ecosystem

Information ecosystems within organizations started with centralized architectures where a single information system would support all processes. However, maintaining this information system was complex and it often ended as a big system with a complex structure. Decentralized information ecosystems emerged as a solution with smaller information systems delineated towards specific processes with data providers and consumers collaborating to enhance interoperability between their information systems. Today, due to conditions such as globalization, multinationalism, these information ecosystems are moving towards highly decentralized architectures. Consequently, there is little to no coordination between data providers and consumers making it difficult to resolve the heterogeneity and enhance interoperability between their information systems. The use of solutions such as data integration or curation to enhance interoperability in these information ecosystems have bring little success due to their high decentralized nature.

To this end, in this chapter, we analyze and demonstrate this assessment in the context of existing information ecosystems, focusing on what we call highly decentralized information ecosystems. Our objective is to understand the problem of heterogeneity and derive the set of requirements to enhance interoperability in the context of highly decentralized information ecosystems. We begin this chapter with Section 1.1 in which we lay down the foundation of highly decentralized information ecosystems. Then, in Section 1.2, we describe the different types of data heterogeneity in a highly decentralized information ecosystem. Finally, in Section 1.3, we study current solutions in the domain of data interoperability that could tackle these problems. From this analysis, we identify the current flaws and define a set of requirements for enhancing interoperability in highly decentralized information ecosystems.

Contents

1.1 Preliminary Definitions

The concept of information system has been studied for decades but still, a consensus has not yet been established on their definitions. Referring to this, [BC15] analyzes thirty-four such definitions while [Pau07] states that "It could be a surprise that what an IS[1] is is not established. On the other hand, since many people are studying IS from a variety of perspectives, maybe it should be no surprise that there are a variety of definitions. But then, how would society know what IS is and what it can do if there is no clear understanding?". To provide a clear and sound frame to our work, in this section, we start with a generic definition of an information system in Section 1.1.1. Then, in Section 1.1.2, we use it as a core notion to define an information ecosystem and its different types to finally finish on highly decentralized information ecosystem.

1.1.1 Information System

An *information system* is *a set of interrelated components that processes data collected or retrieved from its external environment and generates output that can be disseminated to people, stored for later use in a database, sent to itself as a feedback or as input to other systems* [Gro07]. From this definition, we depict an abstract view of an information system in Figure 1.1.

Information System in itself is an abstract notion that can be instantiated in different domains. For example, different types of information systems exist such as management information systems for supporting managerial decision making within organizations or expert systems for emulating decision making ability of human experts. Irrespective of their types, the basic components of an information system include hardware and software components. Hardware components can range from small devices like hard drives and processors to big physical systems dedicated for storage or processing. Similarly, software components can range from minor software systems like inventory controls or accounting systems to complex softwares with different functionalities like ERP (Enterprise Resource Planning) systems.

[1] Information System

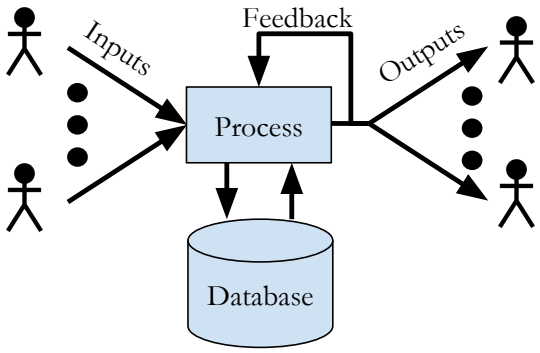

FIGURE 1.1: General overview of an Information System

An information system does not operate in isolation. Instead, it interacts with other actors within an environment that we refer to as an *information ecosystem*. Let us now analyze information ecosystems and their different types.

1.1.2 Information Ecosystems

The concept of information ecosystem has been defined differently in some works [Jac04, CJ06, Bro10] but we provide a unified definition for use in our work. We define it as *a system consisting of actors such as information systems, people (administrators, developers, end-users, etc.) as well as information ecosystems themselves that produce and consume data and may collaborate with each others to satisfy some global objectives.*

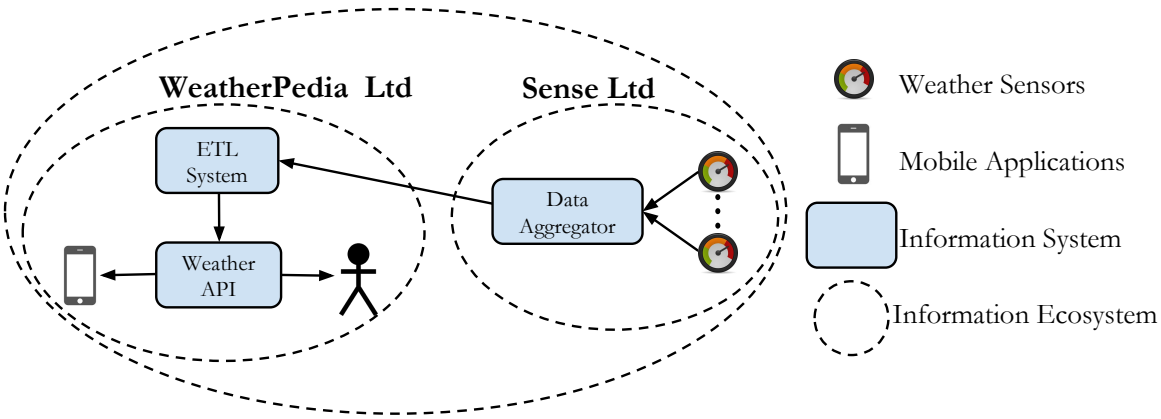

FIGURE 1.2: Example of Information Ecosystems

Let us illustrate this definition using the example shown in Figure 1.2 of an information ecosystem involving two enterprises: `Sense Ltd` and `WeatherPedia Ltd`, each having their own information ecosystem. `Sense Ltd` consists of an information system that collects data from weather sensors deployed at different geographical locations. `WeatherPedia Ltd` consists of two information systems: `ETL System` and `Weather API`. `ETL System` processes data from the information system in `Sense Ltd` and sends its results to `Weather API` which in turn is exposed by the latter as via an API that is used by different actors. Also, the information ecosystems of the

two enterprises interact in a *higher level information ecosystem* which can be due to a partnership or supply chain.

Obviously, the commonplace applications of information ecosystems are in organizations. Two main types of organizational information ecosystems namely intra-organizational and inter-organizational can be identified. An intra-organizational information ecosystem is a system where different actors from an organization produce and consume data to support the organization objectives. Compared to the latter, an inter-organizational information ecosystem consists of actors from more than one organization may be collaborating for similar or different objectives.

Information ecosystems may have a centralized and decentralized architecture [LSH03]. Two concepts that are core to these two architectures are integration and interoperability as discussed in [MPC+07]. While interoperability refers to autonomy and coexistence, integration goes beyond interoperability and involve a degree of functional dependence.

Let us use these two concepts with respect to centralized, decentralized and highly decentralized information ecosystem and their application in the context of intra and inter organizations.

Centralized Information Ecosystem

A centralized information ecosystem *is an information ecosystem consisting of one or more physical and logical systems between which tight relationships have been established to form a family of integrated system that gives the illusion of a single coherent information system built and maintained under the supervision of a central authority that strives to ensure uniformity throughout the ecosystem* [Hug07, PW16].

Normally, in a centralized information ecosystem, the central authority governs every aspect related to the information system that is build upon the different physical and logical systems. As a result, at the time of operation, there is complete integration between the latter systems and no heterogeneity is present.

Now, let us consider the use of a centralized architecture in intra-organization and inter-organization information ecosystems.

Intra-organization information ecosystem Such an information ecosystem with a centralized architecture would mean that a single information system supports all the business needs of an organization. In this case, the central authority would be people such as the IT manager and/or developers from the organization. The information system would consist of several logical parts for managing different areas of the organization and would run on a physical system made up different parts (e.g. processors, disks, network). Tight integration between the different parts enhances standardization and eliminates heterogeneity [WJ06]. However, disruptions (e.g. database failure) in the family of integrated systems may cause losses in significant functionalities and affect all actors in the information ecosystem [MPC+07].

Inter-organization information ecosystem Such an information ecosystem with a centralized architecture would mean a single information system used by all collaborating organizations. In this case, the central authority may consist

of members from each collaborating parties. It needs to take decisions about the different aspects of the system considering the collaborating parties. These decisions may include legal decisions that may be affected by the law of countries in which the parties resides. Moreover, there may be technical aspects such data formats to use or resource allocation to the different parties. Taking these decisions may be difficult due to constraints of the collaborating parties and this may create disagreements between members of the central authority or the parties themselves [Goe08]. In short, setting up such information systems is complex and there are cases where organizations have ceased such efforts due to failure of obtaining a return on their investment [RM13]

Decentralized Information Ecosystem

A decentralized information ecosystem is *an information ecosystem consisting of distinct information systems that may evolve independently while allowing others to use their functionalities together with a coordination authority that governs data interactions between the different information systems* [Hug07]. More precisely, in such an information ecosystem:

- the units managing the information systems can take up two roles vnamely data provider and data consumer;
- data consumers may request data providers for data;
- data providers must respond to requests from data consumers to establish data interactions between their information systems;
- the information systems of data providers and data consumers may not be directly interoperable as they may be heterogeneous;
- data providers and data consumers may negotiate and go through a handshaking process to make their information systems interoperable enough to exchange data;
- while data providers may change their information systems to adapt to dynamic conditions, doing so should not affect the interoperability existing between their information systems and that of their data consumers;
- the coordination authority governs data interactions between the different information systems.

Compared to the central authority in centralized information ecosystems, the coordination authority only govern the data interactions between information systems. More precisely, it may help to set up, maintain and preserve these interactions and resolve conflicts that may occur during these processes. As a result, different units are free to design and take their own decisions with respect to their information systems as long as they welcome new data interactions and preserve existing ones.

Now, let us now consider the use of a decentralized architecture in intra and inter organization information ecosystem.

Intra-organization information ecosystem Such an information ecosystem with a decentralized architecture would mean the existence of several distinct information systems independently built and maintained. Each of these information systems is delineated to support only one area of responsibility (e.g. human resource management) of the organization while at the same time interacting with each other to exchange data when necessary [Hug07]. Using a decentralized architecture in such a context enhances robustness as failure of an information system affect only one area of responsibility. However, the information systems may be based on different technology stacks (e.g. operating systems, network infrastructure) due to being built and maintained independently. Consequently, there may be much heterogeneity (e.g. network, data) between them hindering their ability to interoperate. [WJ06]

Inter-organization information ecosystem Such an information ecosystem with a decentralized architecture would mean that collaborating organizations can set up direct interactions between their information systems without involving everyone. Also, the absence of shared system as in centralized architectures means that there is less concerns about high-level restrictions that may motivate more interactions [Goe08]. As in the previous case, there may be much heterogeneity between the information systems. However, in this case, resolving this heterogeneity may be more complex due to resource constraints (e.g. financial, human) of the collaborating parties.

Highly Decentralized Information Ecosystem

The concept of highly decentralized architectures has been used [WJ08, GSB16, HK18] in the context of information systems but to our knowledge, it has not been explicitly defined and its difference with decentralize architectures is not clear. To this end, we define a highly decentralized information ecosystem as a *decentralized information ecosystem consisting of information systems managed by units that are self-governed and exist with little to no coordination between them.*

In decentralized information ecosystem, the coordination between the different units was helping them to collaborate, set up data interactions between their information systems and exchange data. However, in highly decentralized information ecosystem, a coordination authority may either not exist or if it does, it may not be able to enhance coordination in the information ecosystem due to different reasons based on the context that we will see below. The self-governance of individual units and the lack or absence of coordination create several problems.

Firstly, the different units may work in isolation leading to a problem known as the *information problem* [Fox96] that arises when information systems within an information ecosystem "erect barriers between themselves thereby reducing information flow" [Fox96, Gia04]. As a result, data providers and consumers may be unaware of each other. Secondly, data providers can expose their data sources without considering the capabilities or requirements of data consumers. Consequently, data consumers may face much difficulty when exploiting these data sources. Finally, while data consumers may request for data interactions from data providers

for enhancing interoperability between their information systems, there is no guarantee that these interactions will be considered by data providers. Even if they are considered and implemented, there is no guarantee that they will be preserved if data providers change their information systems.

Normally, in highly decentralized information ecosystem, one approach to tackle the latter problems between data providers and consumers is to introduce data publishers as a third role. Data publishers acts between data providers and consumers and facilitate communication between them using some data platforms. While both data providers and publishers provide data to data consumers, there is a major difference between them as pointed out by Data.gov.uk[1]. Data providers create and own the data and supply it to data publishers along with the required metadata that the latter publishes to data consumers [Mar12]. Data publishers can have a great influence as they become the main access point to discover and exploit data sources. If they do not publish the data properly, data consumers may continue to face difficulty when exploiting the data to the extent that it may be left unexploited.

Intra-Organizational Information Ecosystem In this context, the increasing number of autonomous information systems may enhance the information problem. Moreover, it may be difficult to enhance coordination between them even if a coordination authority exists. In some cases, the number of autonomous information system is in the order of hundreds or even thousands [RV16]. To cite some examples, [FKA+12] mentions that Daimler still has 3000 after a consolidation efforts while Volkswagen has 5000 and in both cases, the information systems are autonomous and heterogeneous. In such conditions, even with the presence of a coordination authority, if a data provider receives request for data interactions from only ten percent of data consumers in the information ecosystem, the latter may not be able to manage due to a resource bottleneck. Moreover, it may be nearly impossible for data providers to consider the requirements of each and every data consumer when exposing their data sources.

Inter-Organizational Information Ecosystem In this context, the lack of coordination between the different organizations may be associated with non-technical aspects such as organizational or political issues [GS16]. Besides this, strategic and cultural incompatibilities [RM13], lack of resources (e.g. financial, human) [KMMV07] and lack of commitment and willingness [Wil07] may also be contributing factors.

Intra/Inter-organizational information ecosystem may initially not be highly decentralized. Before this, they may start as being decentralized and then undergo changes like we saw, increase in autonomous information systems or decreased coordination, that ultimately make them highly decentralized. One indication that current intra/inter-organizational information ecosystems are moving towards highly decentralized architectures is their prevalent use of data curation systems under the banner of enterprise portals to facilitate communication and data interactions between data providers and consumers [DW05a, DW05b, HTD09]. A

[1] https://data.gov.uk/, last accessed on 2 May 2018

2006 study in the US by Forrester Research Inc. showed that 46% of large companies are using an enterprise portal while another quarter intended to use it by 2008 [For06, USR10]. As such, we may presume that by now, the use of enterprise portals may already be a norm in large organizations.

The Web The Web is probably the best example of a highly decentralized information ecosystem. Billions of data publishers and consumers seamlessly exchange data using heterogeneous systems without even knowing the existence of each other. This is possible mainly because of the Web's bottom-up architecture where both data publishers and consumers actively participate by adopting open standards at every level. Web standards and technologies enable data publication in a way that give the illusion of a global repository from which data consumers can browse and "integrate" data seamlessly.

Open Data Context Open data is data that is publicly published to everyone, free, and can be consumed by third parties for their own use or republished under different forms. Adherence to this philosophy creates an information ecosystem, the *open data context*, consisting of actors that publish data publicly or consume openly available data. Such an information ecosystem may be considered as an extended inter-organizational information ecosystem in which organizations as well as the general public may also provide and consume data.

The open data context may be the most common and apparent example of a highly decentralized information ecosystem where collaborating parties are geographically dispersed and self-governed without any coordination authority. Data providers and consumers barely know each other and there may be no direct communication between them. Even if there was a possibility of opening a direct communication, data providers may not participate as no formal contract bindings between them and data consumers.

In the open data context, data publishers play an important role in solving the information problem by providing open data portals to curate data sources from data providers. Normally, these data publishers uses the Web as a platform to instantiate these data portals that have become the main entry point to access open data [UNP15]. Some example of data curation systems are CKAN [1], Socrata [2] and OpenDataSoft [3].

1.2 Heterogeneity Problems

In centralized information ecosystem, at the time of operation, all heterogeneities would have been solved through standardization and tight integration between the physical/logical systems [WJ06]. However, in (highly) decentralized information ecosystem, heterogeneity may still prevail while the information systems are in operation thus preventing them to interoperate and exchange data. Several types of heterogeneities (e.g. of networks, operating systems, programming languages, etc.)

[1] https://ckan.org/, last accessed on 2 April 2018
[2] https://socrata.com/, last accessed on 2 April 2018
[3] https://www.opendatasoft.fr/, last accessed on 2 April 2018

may exist due to the varying technology stacks on which information system have been implemented. Normally, they may be abstracted through the use of high-level data access systems such as web services [MBB+03, SVVB12, QLS+11]. As a result, data heterogeneity is the only heterogeneity that remains to be handled to enhance interoperability between the different information systems.

There is no universally accepted definition of data heterogeneity [MBWdll15]. Instead, some existing definitions are even conflicting. To cite only some examples, [Kin08] defines data heterogeneity as consisting of syntactic, semantic and structural heterogeneity while [VSS02, Gag07] make no difference between data heterogeneity and semantic heterogeneity. To avoid these inconsistencies, we provide our definition of data heterogeneity that is actually extrapolated from the definition of data interoperability, that will discussed in Section 1.3.2, as they are closely linked to each other. For example, data heterogeneity can be seen as a problem to data interoperability but data interoperability can also be seen as a solution to data heterogeneity.

To this end, we define data heterogeneity as *the differences that prevent data to be universally accessible, reusable and comprehensible by all transaction parties (in a human-to-machine and machine-to-machine basis) caused by the use of different representations, different purposes, different contexts, and different syntax-dependent approaches.* Based on this definition, several dimensions of data heterogeneity can be identified namely at the data access, syntactic, semantic and structural level.

In the remainder of this section, we define the dimension of data heterogeneity with respect to data access (Section 1.2.1), syntax (Section 1.2.2), semantics (Section 1.2.3) and structure (Section 1.2.4) and illustrate them using an example. Also, we relate the definitions to parts of the data heterogeneity's definition using italicized text

.

1.2.1 Data Access Heterogeneity

Data access heterogeneity may occurs as a result of providing different kinds of access to their data raising heterogeneities that relate to data location [BKLW99] and data view [Jau99], data organization [RNC+03]. [CKRJ17] extends these heterogeneities to include data sources' varying querying capabilities including their syntax and semantics.

Based on the above, we define data access heterogeneity as *the heterogeneity raised when accessing data at different data sources.* With respect to our definition of data heterogeneity, data access heterogeneity prevents data from being *universally accessible.* Data access heterogeneity includes:

- *location heterogeneity*: data may be accessed at different data sources with different locations;
- *view heterogeneity*: different views of data may be provided at different abstraction levels using different data organization schemes;
- *data access protocol heterogeneity*: different protocols having varying querying capabilities, syntaxes and semantics may be used to access data.

Let us now consider an example to illustrate data access heterogeneity. Suppose that a data consumer is in need of some data. The first step is to identify the data sources that are available in the information ecosystem and their locations. To facilitate this process, platforms such as data curation systems that will be discussed in Section 1.3.2 may be used. After having determined the location of their data access system, data consumers may need to tackle data access protocol heterogeneity and view heterogeneity as data curation systems may be using different protocols and views. For example, as mentioned before, CKAN and OpenDataSoft are two different data curation systems widely used in the open data context and both have their own custom web API that provide access to their data [KRT+16]. While both web APIs are based on HTTP, they have different syntaxes for HTTP requests such as using different headers or URL parameters. Also, data consumers may need to deal with data view heterogeneity as different data access systems may provide different views and organization of their data. Again, considering CKAN and OpenDataSoft, both provide different views of their data by using different endpoints [CZdS18].

In summary, the problem raised by data access heterogeneity is that data consumers cannot use a single process to consume data from several data access system as they need to cater for their different technicalities. Also, if metadata about the data access protocol and view is not provided, data consumers may face much difficulty when using the data access system.

1.2.2 Syntactic Heterogeneity

Syntactic heterogeneity has been defined in different ways. In some cases, it has been attributed to structural heterogeneity [MZG14, BWF13] or even hardware and software heterogeneity [HMTF09]. In our case, as we mentioned before, our focus is on data. Also, there is a clear distinction between a data structure and a data syntax as different structures may be encoded in the same syntax. But what we find common in many definitions of syntactic heterogeneity is the mention of data formats and data syntaxes [Car07, HTP+09, JSC18] that aligns well to our definition of data heterogeneity.

Therefore, we define syntactic heterogeneity *as the heterogeneity caused by the use of several data syntaxes or formats*. With respect to our definition of data heterogeneity, it may occur due to syntax-dependent approaches in legacy systems. For example, the XML syntax was prevalent during the late 90s and therefore interoperating with information system that use them today may raise the syntactic heterogeneity if data syntaxes other than XML are used such as JSON.

Let us now consider an example of syntactic heterogeneity. After having resolved the data access heterogeneity discussed in the previous section, data consumers may finally obtain a copy of the data in their environments. However, it may not be directly exploitable as the data may be in different syntaxes. Let us illustrate this problem using an example of data from two different sources shown in Listing 1.1 and Listing 1.2 that shows part of documents containing details of parking facilities in two different documents.

Listing 1.1: Data in XML

```
<ParkingFacilities>
 <CarParking>
   <id>50</id>
   <space>5</space>
   <lat>48.845</lat>
   <long>2.373</long>
 </CarParking>
...
</ParkingFacilities>
```

Listing 1.2: Data in JSON

```
{ "ParkingLots": [{
    "code": "CP:50",
    "type":"CarParking",
    "space": 5,
    "lat": 48.84524,
    "long": 2.37322,
    },...]
  }
}
```

As we can see, the two documents are encoded in the XML and JSON syntax respectively. As such, they cannot be decoded using the same process as they are in different syntaxes. This problem is usually considered to be the simplest as it may be resolved by automatically encoding all the data in a single syntax [MPS10]. However, even if similar syntax is used for both documents, they may not be directly exploitable as they are still structured using different terms and models.

1.2.3 Semantic Heterogeneity

Semantic heterogeneity is considered to be the toughest among the dimensions of data heterogeneity [MBB+03, LKdL15, FMW+14, Oli17]. While its broad definition seems to be stable in the literature [HTP+09, NVS+06], there is some disagreements in its constituents. For example, [Bis98] states that semantic heterogeneity can be divided into cognitive heterogeneity and naming heterogeneity while [XL02] states that it may also include formalization heterogeneity, conceptualization heterogeneity and context heterogeneity. However, we restrict our definition to a broader level as differentiating between these constituents are not the focus to this work.

Therefore, we define semantic heterogeneity as *differences in lexicals that yield different meanings and interpretations*. More specifically, different lexicals may have the same interpretation or similar lexical may have different interpretations. With respect to our definition of data heterogeneity, it mostly occurs when data is generated in different *contexts* by independent parties [Hal05]. Its occurrence prevents data to be *universally comprehensible and reusable*.

Let us now consider an example of semantic heterogeneity. Consider the term space from Listing 1.1 and Listing 1.2 respectively. The documents from which these terms emerge may have come from different data providers. Intuitively, we may think that both terms refer to the number of car parking spaces in a parking facility. However, now suppose that the JSON document comes from a real-time data source. In the latter case, ttextttspace may instead refer to the remaining number of parking spaces. In short, we cannot be sure of the term's interpretation as long as we do not have its explicitly interpretation from the right source (e.g. metadata supplied by data provider). Without this, we can only rely on our common sense and doing so is considered to be problematic when resolving semantic heterogeneity [HG01]. Unresolved semantic heterogeneity may generate invalid result that may be left unnoticed if users are unaware of the semantics of the terms [HG01].

1.2.4 Structural Heterogeneity

Structural heterogeneity has been attributed to differences in data stuctures [AN15, MO04, NVS+06]. Schematic heterogeneity appearing in structured databases where schemas have a prime importance is also an aspect of structural heterogeneity [She01]. [NVS+06] defines model/representational heterogeneity separate from structural heterogeneity but we do not make such difference. Structural heterogeneity may be caused by different views on the same domain by independent parties [PCDT05]. It may also be be linked to syntactic heterogeneity because data syntaxes may place constraints on the structure of data encoded [NJN, FL11]. For example, XML dictates a hierarchical structure in the data with a root element.

We define structural heterogeneity as *the heterogeneity caused by the use of several data representations, structures, domain models or schemas*. With respect to our definition of data heterogeneity, structural heterogeneity may occur when data structures are dependent on *context* and *purpose*. Thus, when similar data from different `context` or generated for different `purposes` may not be reused directly due to their varying structures. Thus, structural heterogeneity may make data less *reusable*.

Let us consider an example of structural heterogeneity using the documents in Listing 1.1 and Listing 1.2. While their domain models have not been explicitly given, some structural differences may be visible in the data itself. By looking at Listing 1.1, we can intuitively infer that the first object in the element `ParkingFacilities` describes a car parking facility because of the `CarParking` element. However, in Listing 1.3, the type of the parking may be known by looking at the `type`. In other words, different structural primitives, classes and attributes, may have been used to describe the domain objects.

Listing 1.3: Data in XML

```
<ParkingLots>
 <Parking>
   <id>50</id>
   <type>car</type>
   <space>5</space>
   <lat>48.845</lat>
   <long>2.373</long>
 </Parking>
...
</ParkingLots>
```

To overcome structural heterogeneity, a pivot model may need to be used. Suppose, the model in Listing 1.1 is used as the pivot model for storing parking facilities' details. Mappings may be established between terms from the model in Listing 1.1 and Listing 1.2. However, before this, semantic heterogeneity should be resolved due to differing interpretation of terms. For example, mapping `ParkingLots` to `ParkingFacilities` require that their interpretation are known to be the same. The latter case is independent of structural heterogeneity. However, semantic heterogeneity may be complexified by structural heterogeneity and their combination is considered to be the toughest form of heterogeneity [RMB+12]. For example, mapping `space` from Listing 1.3 to `space` is possible only if in the former case, the `space` element occurs in a `CarParking` element.

Overall, structural heterogeneity may make information retrieval more complex.

For example, both the structure and terms used in queries may be dependent on the domain model. Therefore, the same query may not be used on data sources having different models. More abstractly, a single process cannot be applied to precisely manipulate or extract data from data sources having semantic and structural heterogeneity.

1.2.5 Synthesis

As we have seen, the different dimensions of data heterogeneity make data exploitation complex. There is not much difference between how data heterogeneity occurs in decentralized and highly decentralized information ecosystem. Yet, data heterogeneity remains a core problem in highly decentralized information ecosystem.

As mentioned in Section 1.1.2, the coordination and collaboration of data providers and consumers enable them to actively interact to resolve data heterogeneity. However, as mentioned in Section 1.1.2, in highly decentralized information ecosystem, data consumers may barely interact with data providers as a result of which they have to tackle the data heterogeneity completely on their own. To resolve the data heterogeneity, data consumers depend on metadata exposed by data publishers. Thus, if data publishers do not use the appropriate standards and technologies to publish the data and metadata from data providers, it may be difficult for data consumers to make sense of the data.

Now that we have seen the problem of data heterogeneity in highly decentralized information ecosystem, let us analyze the different approaches that may be used to enhance data interoperability.

1.3 Data Interoperability Requirements

As we have seen in the previous section, data heterogeneity is a barrier towards achieving interoperability at the data level in (highly) decentralized information ecosystem. In this section, we discuss the interoperability between information systems in Section 1.3.1. Then, in Section 1.3.2, we analyze and criticize data interoperability techniques with the aim of identifying their limitations. Finally, in Section 1.3.3, considering these limitations, we set requirements for a solution to achieve data interoperability in (highly) decentralized information ecosystem.

1.3.1 Data Interoperability

Interoperability "*is the ability of two or more systems or components to exchange information and to use the information that has been exchanged*" [IEE91]. As shown in Figure 1.3, interoperability can take place at different levels. The European Interoperability Framework [Com17] categorizes them in four main levels: legal, organizational, semantic and technical; three of which are shown in Figure 1.3.

At the highest level is legal interoperability that is the ability for organizations to collaborate under different legal frameworks. Then, organizational interoperability enhances mutually benefits through alignment of business processes. Comprehensibility between communicating parties is achieved by semantic interoperability.

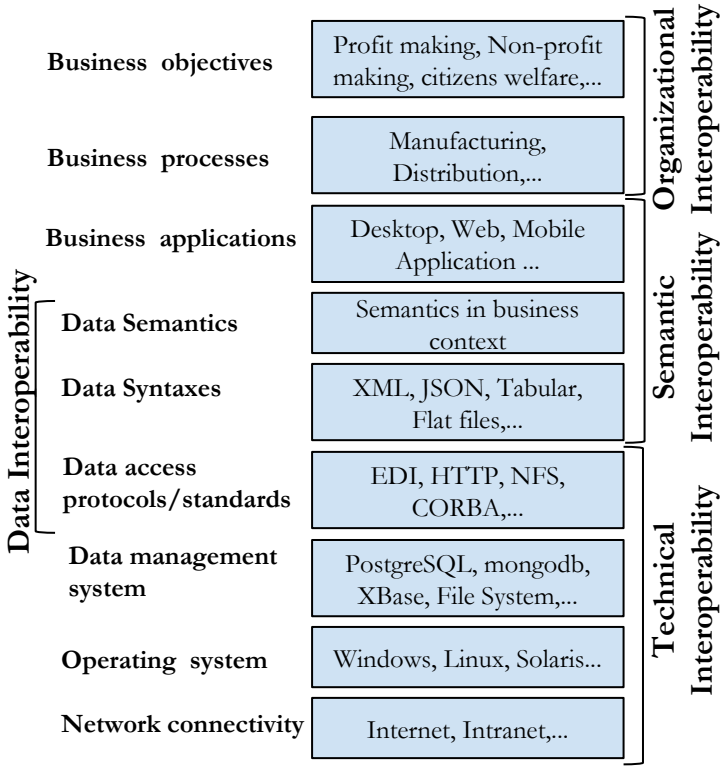

FIGURE 1.3: The Interoperability Stack adapted from [ZD04] and [RCL14]

Finally, technical interoperability enhances communication between systems implemented using different technology stacks (e.g. network infrastructures, operating system).

One way to abstract a subset of layers from the technology stack is to use web services. For this purpose, web services are defined as loosely coupled and platform independent systems [MBB+03] that hide platform heterogeneity (e.g. network, operating system, etc.) to provide seamless data interaction between heterogeneous information systems [SVVB12, QLS+11]. Web services working at the *Data access protocol/standard* may be used to abstract part of the technology stack and enhance seamless data interaction between heterogeneous systems [SVVB12, QLS+11]. Consequently, the only layers left to be dealt are those of data interoperability.

As shown in Figure 1.3, data interoperability consists of semantic and partly of technical interoperability. It can be defined as "*the ability of data (including documents, multimedia content and digital resources) to be universally accessible, reusable and comprehensible by all transaction parties (in a human-to-machine and machine-to-machine basis), by addressing the lack of common understanding caused by the use of different representations, different purposes, different contexts, and different syntax-dependent approaches*" [KLM+11, LKA+12, KML14, GA18].

As mentioned in Section 1.2, data interoperability and data heterogeneity are closely linked. In fact, we consider data heterogeneity to be the main problem in enhancing data interoperability between information systems. The dimensions of

data heterogeneity described in Section 1.2 can be aligned to the layers of data interoperability shown in Figure 1.3. Data access, syntactic and semantic heterogeneity can be aligned to *Data access protocols/standards*, *Data Syntax* and *Data Semantics* layers respectively. As for structural heterogeneity, it may occur both in *Data Syntax* and *Data Semantics* layer because as we mentioned in Section 1.2.4, it is linked to both data syntaxes and semantics.

Let us now consider approaches that can be used to enhance data interoperability in the next section.

1.3.2 Current Approaches to Data Interoperability

Data curation and data integration are two main approaches that may be used to enhance data interoperability. In this section, we discuss them in Section 1.3.2 and Section 1.3.2 with respect to highly decentralized information ecosystem.

Data Curation

Data curation is the process of organizing data with the aim of enhancing data interoperability by providing its consumers with data having a high readability, accessibility and reusability [Yak07]. Data curation system is considered to be a key solution "where there is an increase in the number of data sources and platforms for data generation" [FC16]. They are used both in enterprises and open data context where they have been instantiated as enterprise data portals [Whi00] and open data portals [HVdB11] respectively.

Data curation system may be used to to facilitate the discovery and access of data sources together with the required metadata to exploit them [Haz02, Cha13b, Joh17]. Therefore, they may be used by data publishers to expose data from data providers in highly decentralized information ecosystem.

Problems related to the use of data curation systems As mentioned in Section 1.1.2, in highly decentralized information ecosystems, data publishers may use some data platforms to expose data and metadata from data providers. Data curation systems are one type of platforms they may used to do so. However, at this point, there is one main problem that may be raised. As we mentioned before, since in highly decentralized information ecosystem data providers are self-governed units with no coordination between them, data from them are very likely to be heterogeneous. Consequently, if data publishers directly expose data and metadata from them via data curation systems without using proper standards and technologies, there may be more data heterogeneity. This may be considered as a limitation in the practices of the data publisher. However, it may also be a limitation of the data curation system in use as it may not enforce the use of proper standards and technologies and may not be equipped with tools to facilitate their use.

Thus, while data curation systems may be used to abstract data heterogeneity, naively using them may generate more data heterogeneity than there is already making data integration by data consumers more complex.

Data Integration

Data integration is the process of providing a unified view of data combined and reconciled from diverse and heterogeneous sources to users (people or machine) to facilitate their exploitation [Len02]. It has been the classical solution to abstract data heterogeneity since long [Wie92]. Generally, two main approaches to integrate data exist: point-to-point and mediator [Gia04, KA15].

In the point-to-point approach, data consumers have to explicitly tackle every data source and handle its data heterogeneity. Figure 1.4 (a) shows an abstract view of this approach. It has several disadvantages [LSH03, NPF+13]. This approach is not scalable especially where the number of data sources is increasing rapidly because every data consumer wishing to exploit the data sources has to repeat the same procedure for doing so [KA15]. Moreover, to exploit data sources, data consumers have to define wrappers resulting in their information systems being dependent on low-level details. Consequently, minimal changes in a particular data source may require data consumers to modify their wrappers.

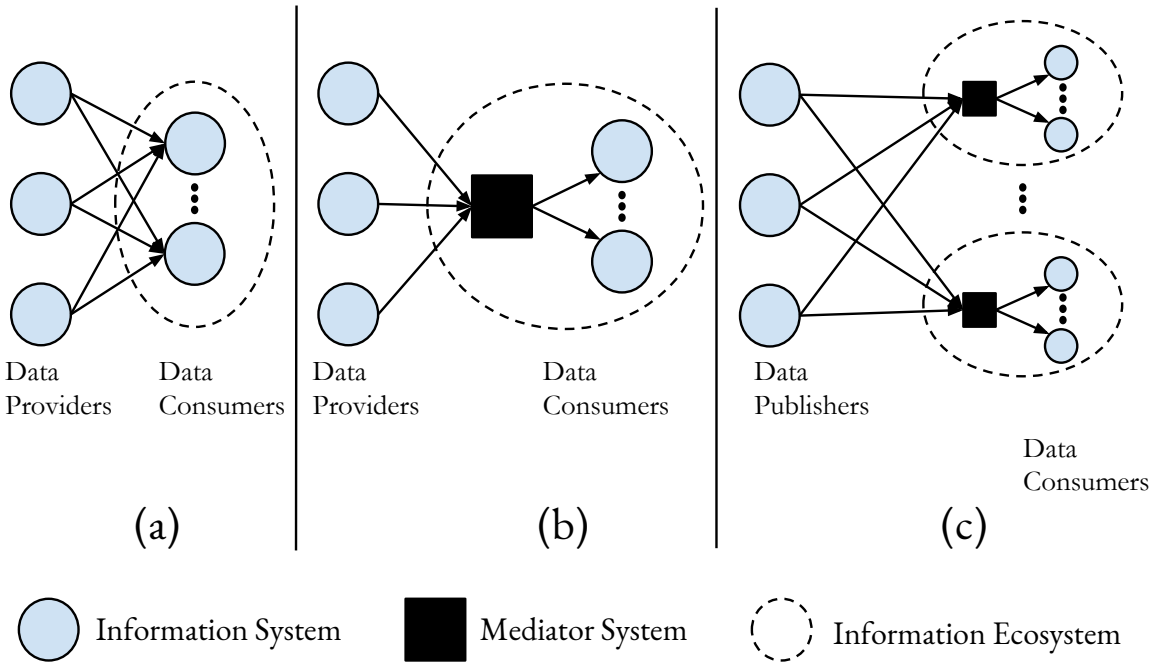

FIGURE 1.4: Data Integration Approches

On the contrary, in the mediator approach [Wie92], as shown in Figure 1.4 (b), an intermediary layer (the mediator) is introduced between the heterogeneous data sources and the information system. The mediator abstracts the heterogeneities provide data consumers with a homogeneous view of data [Gia04]. The data consumers are unaware of the complexity incorporated by the mediator and have the illusion of interacting with a single system. Mediator approaches can be categorized based on different dimensions [KP09] (e.g. data model) based on which different mediator systems have emerged [DD99] with the classical ones being federated databases, data warehouses, mediated query systems and lately data lakes [SM14]. While mediator systems can provide a homogeneous view to data, their use may raise some problems in highly decentralized information ecosystem.

Problems related to the use of mediator systems Figure 1.4 (c) shows an abstract overview of a highly decentralized information ecosystem where data publishers directly expose data and metadata from data providers via some platforms (e.g. data curation systems). Two main problems may be raised when using mediator systems resolve data heterogeneity generated by the platforms..

The first problem is that both the development and maintenance of mediator systems depends on the availability of metadata whose quality can strongly affect their success [SR96, AF05, CZ06, MRW10]. As mentioned before, in highly decentralized information ecosystem, there may be no interaction between data providers and consumers. Thus, data consumers depend entirely on data publishers for obtaining good quality metadata. In the case where data publishers are exposing the metadata directly as they are supplied by data providers, the metadata itself may be heterogeneous, thus affecting its quality and eventually that of the mediator systems.

The second problem is effort duplication. As we mentioned before, in highly decentralized information ecosystem, data consumers are self-governed without any coordination between them and may thus be unaware of each other. Consequently, they may duplicate their efforts to build and maintain mediator systems for exploiting some data sources that may sometimes be even the same. From an abstract perspective, when there are several data publishers as in Figure 1.4 (c), doing so emulates a point-to-point depends on the efforts data publishers. The more effort data publishers do to expose data and metadata using the proper standards, the less effort is required by data consumers during data integration .

In summary, as we can see, the success of data integration heavily depends on how data has been exposed by data publishers. Let us now see in the next section requirements that may be followed by data publishers to facilitate exploitation of data in highly decentralized information ecosystem.

1.3.3 Requirements

As we have seen in the previous section, the success of data curation and integration systems depends on how data is exposed by data publishers. Therefore, it is important to identify the requirements that data publishers need follow to expose data in a way can enhance data interoperability in highly decentralized information ecosystem.

In Section 1.1.2, we described that the Web is the perfect example of a highly decentralized information ecosystem where in spite of the high heterogeneity, different types of systems can still interoperate and exchange data. Thus, requirements to enhance data interoperability in highly decentralized information ecosystems may be derived by considering the Web, its design principles and technologies used on it. However, currently the Web is geared for humans and therefore additional requirements may be needed to enable seamless communication and data integration by information systems. Moreover, since highly decentralized information ecosystem is a superset of decentralized information ecosystem, a solution for the former would be applicable for the latter also.

In the remaining of this section, we identify requirements that a solution for achieving data interoperability in highly decentralized information ecosystem

should satisfy. We group the requirements in data syntax (Section 1.3.3), data semantics (Section 1.3.3) and data access (Section 1.3.3) and in each of these sections, the title of paragraphs correspond to the names of requirements.

Data Syntax

In this section, we discuss requirements that have to be respected in highly decentralized information ecosystems with respect to data syntax.

Standardized Data Syntax HTML is used as the pivot data syntax to encode data in Web pages, thus preventing syntactic heterogeneity. It has a syntax and semantics with respect to data presentation. Since it is an open standard, everyone can implement tools that processes document in this syntax. Similarly, in information ecosystems, automatic data exchanges take place between information systems and therefore, to enhance universal reusability, *there is a need for an open standard that defines a language that may be used as a pivot data syntax* to encode description of resources.

Resource Identification On HTML pages, it is possible to exactly refer to a resource on the web as HTML enforces the use of a syntactic element, the anchor, to use hyperlinks to refer to Web resources using their IRIs. Hyperlinks constitute a core element of the Web and allows IRIs to be used as foreign keys when referring to Web resources. Similarly, in an information ecosystem, *the pivot data syntax should enforce the use of IRIs for referring to resources.*

Flexible Data Syntax Information systems may need to directly merge description of resources when integrating data. The resources may be from different data providers and thus may have varying structures. There are data syntaxes where documents in the same data syntax may not be directly merged. For example, XML documents cannot be directly merged as XML is constrained by XML schemas that sets implicit links between entities. Therefore, *the pivot data syntax should be flexible enough to enable the integration of resources having varying structures.*

Data Semantics

In this section, we discuss requirements that have to be respected in highly decentralized with respect to data semantics.

Globally-scoped Identifier The use of URLs as global identifiers enables unambiguous referencing of Web resources throughout the Web. In highly decentralized information ecosystem, local identifiers for resources may be used by data providers within their information systems. Directly exposing the resources externally may create ambiguities. For example, two resources with similar local identifiers may identify different things. Therefore, *resources should be named using identifiers that are globally-scoped at least within the boundary of the information ecosystem.*

Ontology Language On the Web, when humans read some description in natural languages, they may encounter unknown terms and use external sources such as dictionaries to find their semantics. Similarly, when data consumers integrate data that originate from different data providers, they may have to deal with unknown terms. To achieve universal comprehensibility, *there is a need for a formal language to define ontologies that provide explicit semantic of terms.*

Semantics In Syntax When autonomous information systems exchange data, the semantics of terms need to be encoded with the data to make it self-descriptive. Therefore, irrespective of the formalism used to encode terms' semantics, *it should be possible to encode the semantics of terms in the data syntax.*

Data Access

In this section, we discuss requirements that have to be respected in highly decentralized with respect to data access.

Standardized High-level Data Access Web hyperlinks refer to resources and on following them, web browsers communicate with web servers to download the resources. The implementation-specific details of web browsers and servers do not affect the latter communication as the use of high-level data access systems based on different standards (e.g. HTTP, FTP) abstracts the required heterogeneities. Similarly, between information systems in highly decentralized information ecosystems, there is a need for *a high-level data access standard to enable seamless data interactions.*

High-level Data Access Description On the Web, different types of high-level data access systems (e.g. websites, web services) exist. Descriptions of the latter systems are primordial to enable their exploitation. For example, ProgrammableWeb curates thousands of web services and provides links to their documentation wherever possible. In the case of websites, descriptions in natural language allow users to find their way. Similarly, in highly decentralized information ecosystems, *descriptions of high-level data access systems should be provided* to facilitate their exploitation.

1.4 Summary

In this chapter, our objective was to come with a set of requirements of achieving interoperability between heterogeneous information systems in highly decentralized architectures. So, we have started by laying down the core foundations, namely information systems, the environment in which they operate that we refer to as an information ecosystem and their different types. We have seen that in (highly) decentralized information ecosystem, there heterogeneous information system that may need to interoperate for collaborate to achieve some objectives. To enable their interoperation, data heterogeneity is the main obstacle. While in decentralized information ecosystem, it can resolved using data integration, in highly decentralized information ecosystem, this is not possible due to a lack of metadata caused by the information problem. While trying to address the lack of metadata, data

curation may introduce more heterogeneity in the information ecosystem and therefore may not be the right solution. Finally, making analogy to the Web wherever possible, we have set requirements that a solution to achieve interoperability in highly decentralized information ecosystem must fulfill.

Chapter 2

Semantic Web Technologies

In the previous chapter, we have identified requirements with respect to data syntax, data semantics and data access for enhancing data interoperability in highly decentralized information ecosystem. Standards and technologies that were originally conceived for the Semantic Web can be of great use to address these requirements by enabling the creation of information ecosystems where data can be shared using a flexible data model with well-defined semantics using standard data access mechanisms [Wu01, SAD+15]. Consequently, data can be shared and reused unambiguously across applications, enterprises and enterprise boundaries [Lie16].

In this chapter, our objective is to analyze how Semantic Web standards and technologies can satisfy the requirements for data interoperability in highly decentralized information ecosystem. To this end, we describe Semantic Web standards and technologies and explain how they satisfy requirements for data interoperability with respect to data syntax, data semantics and data access in Section 2.1, Section 2.2, Section 2.3 respectively. Finally, in Section 2.4, we describe how Linked Data Platform, one among the Semantic Web technologies, can be used to provide a standard access to data and show the limitations of tools in automatizing the generation of such platforms.

Contents

2.1 Data Syntax

In this section, we explain how the Resource Description Framework(RDF) [CWL14], the underlying data model of Semantic Web, and its implementations can be used to enhance data interoperability at the syntactic level in highly decentralized information ecosystem. To this end, in Section 2.1.1, we describe the basics of RDF and explains how it satisfies the requirements with respect to data syntax described in Chapter 1 (Section 1.3.3). Then, in Section 2.1.2, we describe some implementations of RDF.

2.1.1 RDF

RDF enables asserting statements about resources, known as RDF statements, in expressions known as RDF triples. An RDF triple consists of a subject, a predicate and an object. The predicate expresses a binary relationship between the subject and object. An RDF triple can be seen as an edge $subject \xrightarrow{\text{predicate}} object$ and joined with other triples to create a graph-like data model. A set of RDF triples makes up an RDF graph. A graphical illustration of an RDF graph is also shown in Figure 2.1.

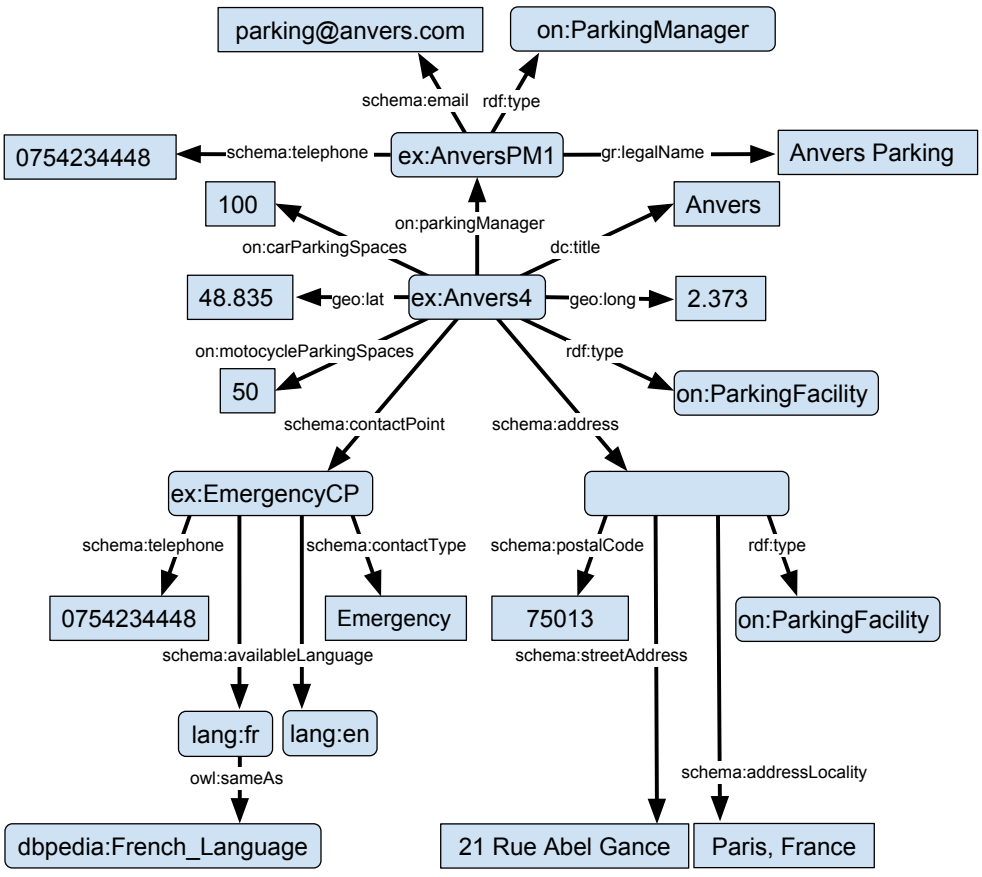

Figure 2.1: RDF Graph Example

In Figure 2.1, rounded rectangles with prefixed IRIs represent resources identified using these IRIs, rectanges represent literals and empty rounded rectangles represents blank nodes that are discussed below. In this graph, labeled arrows represented predicates and their origins and destinations represented subjects and objects of RDF triples.

Listing 2.1 shows the serialization of this RDF graph in the Turtle syntax [BBL08]. There are several syntaxes (e.g. XML, JSON) in which RDF graphs can be serialized but we only use the Turtle syntax. We use both the RDF graph and its prefix declarations throughout this chapter.

```
1 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
2 @prefix owl: <http://www.w3.org/2002/07/owl#>.
3 @prefix dc: <http://purl.org/dc/terms/>.
4 @prefix ex: <http://example.org/data/>.
5 @prefix on: <http://example.org/ontology/>.
6 @prefix geo: <http://www.opengis.net/ont/geosparql#>.
7 @prefix schema: <http://schema.org/>.
8 @prefix lang: <http://id.loc.gov/vocabulary/iso639-1/> .
9 @prefix gr: <http://purl.org/goodrelations/v1#>.
10 @prefix dbpedia:<http://dbpedia.org/resource/> .
11 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
12 @prefix mobivoc: <http://schema.mobivoc.org/>.
13 ex:Anvers4 rdf:type on:ParkingFacility;
14   dc:title "Anvers";
15   geo:lat 48.83523833495664;
16   geo:long 2.37322158810141;
17   on:carParkingSpaces 100;
18   on:motocycleParkingSpaces 50;
19   on:parkingManager ex:AnversPM1;
20   schema:contactPoint ex:EmergencyCP;
21   schema:address [
22     a schema:PostalAddress;
23     schema:streetAddress "21 Rue Abel Gance";
24     schema:postalCode "75013";
25     schema:addressLocality "Paris, France";] .
26
27 ex:EmergencyCP schema:contactType "Emergency";
28    schema:availableLanguage lang:fr,lang:en;
29    schema:telephone "0754234448" .
30 lang:fr owl:sameAs dbpedia:French_language .
31
32 ex:AnversPM1 a on:ParkingManager;
33   gr:legalName "Anvers Parking";
34   schema:telephone "0754234448";
35   schema:email "parking@anvers.com" .
```

Listing 2.1: RDF Graph Example in Turtle

Now, let's further describe RDF with respect to the requirements of data syntax defined in Chapter 1 (Section 1.3.3).

Standardized Data Syntax RDF contributes to this requirement by being an open standard describing a data model that can be used as a data syntax. Also, there are accompanying standard specifications such as [GS14] and [SKL14] that describe ways of serializing RDF graphs in commonly used data formats XML and JSON respectively. Consider the RDF graph in Listing 2.1, its serialization in XML and JSON are available in Appendix A.

Resource Identification RDF contributes to this requirements by enforcing Internationalized Resource Identifiers (IRI) [DS05] as names for resources. Using IRIs, resources can be referenced irrespective of their location. They may be described in the RDF graph where the reference is made or in external data stores. For example, the referenced resource `lang:fr` on Line 28 (Listing 2.1) is actually described and hosted in a different data store.[1] In cases where IRIs may be used, blank nodes, considered as an existential variable, may indicate the presence of a resource without specifically identifying it. In Turtle syntax [PC14, §2.6], blank nodes may be denoted using `[]`. For example, we use a blank node (Listing 2.1, Line 22 - 25) because we do not want to specifically identify addresses of parkings.

Flexible Data Syntax XML or relational data model require predefined explicit schemas. Also, data in these models may not be automatically merged without considering their semantics and naive merging mechanisms may not yield valuable results. On the contrary, RDF requires no predefined schema but instead RDF data itself contains an implicit schema that is automatically extended by adding new predicates or when merging RDF graphs having different implicit schemas. Consequently, this contributes to the flexibility of RDF and allow RDF-based systems to cope with data having varying schemas.

As we have seen, RDF contributes to all the requirements for enhancing data interoperability at the syntactic level. Let us now discuss some of its implementations that facilitates its use.

2.1.2 RDF Implementations

Different categories of RDF implementations exists. Frameworks for facilitating the use of RDF exist in different programming languages. For example, RDFLib[2] and Apache Jena[3] are RDF frameworks in Python and Java respectively. Also, there are editors for facilitating the editing of RDF documents. For example, OpenLink RDF Editor[4] is an RDF editor that eases writing RDF graphs by exploiting ontologies (discussed in Section 2.2) and IsaViz[5] is graphical editor for authoring RDF graphs.

In order to deal with data heterogeneity that is part of our evaluation criteria as mentioned in the Introduction of this thesis, RDF conversion is an important concern to consider in this work. RDF implementations that may be used for doing so are called RDF converters. Some of RDF converters are targeted to data in specific formats while others are more open.

GRDDL [Con07] and XSPARQL [AKKP08b] were originally conceived for extracting RDF data from XML data. GRDDL [Con07] is an approach of assigning transformations (normally in XSLT) to XML documents that may then be used to generate RDF. XSPARQL [AKKP08b] extends XQuery and allows defining transformations from XML to RDF and vice versa. Extensions have been made in it

[1] `http://id.loc.gov/vocabulary/iso639-1/fr.html`, last accessed on 26 July 2018
[2] `https://rdflib.readthedocs.io/en/stable/`, last accessed on 14 September 2018
[3] `https://jena.apache.org/`, last accessed on 14 September 2018
[4] `http://osde.openlinksw.com/`,last accessed on 14 September 2018
[5] `https://www.w3.org/2001/11/IsaViz/`,last accessed on 14 September 2018

for querying relational databases [LBDP11] and data encoded in the JSON format [DPLB14].

Direct Mapping [ABPS12] and R2RML [DSC] are W3C recommendations for transforming relational databases to RDF or mapping queries in SPARQL against virtual RDF views into SQL queries. Direct Mapping defines specific mappings to RDF Graphs using different primitives such as tables, primary keys or joint relationship from the relational data. Consequently, the output RDF graph reflects the structure and vocabulary of the original data's schema. R2RML employs a more generic approach by providing a language to define customized mappings in which the structure and target vocabulary are chosen by the mapping author.

RML [DVSC+14] and SPARQL Generate [LZB17b] are languages that can consider data sources in different formats and languages. RML extends R2RML with language constructs for considering considering new types of data sources (e.g. CSV, XML). Thus, its model requires modification for coping with newer the types of data sources. SPARQL Generate is an extension of SPARQL and uses the extension mechanism of SPARQL to deal with new types of data sources. Thus, in an implementation of SPARQL Generation, dealing with new data formats merely consist of developing new functions conforming to the existing SPARQL specifications.

As seen in this section, RDF with the accompanying tools for converting different data formats to it provides a viable approach for tackling with data heterogeneity at the syntactic level. However, usage of RDF, either natively or when converting from heterogeneous data sources, require ontologies to describe data semantics that we consider in the next section.

2.2 Data Semantics

Besides a data model, RDF also comes with a built-in vocabulary, the RDF vocabulary, that can be used to describe simple cross-domain aspects but lacks the required expressivity to represent complex knowledge. Therefore, in this section, we describe Semantic Web standards RDF Schema (RDFS) [BG14] and Web Ontology Language (OWL) [W3C12] that are sufficiently expressive to represent complex knowledge. Also, we explain how they can enhance data interoperability at the semantic level in highly decentralized information ecosystem. In the remainder of this section, we describe RDFS and OWL in Section 2.2.1 and Section 2.2.2 respectively. Then, in Section 2.2.3, we describe the management of RDFS/OWL ontologies

2.2.1 RDFS

RDFS provides a data-modeling vocabulary to enable the definition of ontologies. RDFS extends the RDF vocabulary and is therefore directly written in RDF. From terms in the RDFS vocabulary, five properties are key ones namely `rdf:type`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain` and `rdfs:range` [Hog14]. `rdf:type` is used for specifying resources as instances of some classes. `rdfs:subClassOf` and `rdfs:subPropertyOf` allows defining hierarchies of classes and properties. `rdfs:domain` of a property specifies the class of the subject in triples where the property is used as a predicate. Similarly, `rdfs:range` of a property specifies the class or datatype of the object in triples where the property is used as a predicate.

Listing 2.2 shows an RDFS ontology. It uses `rdfs:subClassOf` to indicate a subclass relationship between `on:ParkingManager` and `gr:BusinessEntity` (Listing 2.2, Line 9) and `rdfs:subPropertyOf` to indicate a sub-property relationship between `mobivoc:entrance` and `dc:hasPart` (Listing 2.2, Line 15). Also, it uses `rdfs:domain` and `rdfs:range` to indicate that the domain and range of `on:parkingManager` is `on:ParkingFacility` and `on:ParkingManager` respectively.

```
1 <http://example.com/ontology> rdfs:label "An ontology for describing
      parking facilities"@en .
2
3 on:ParkingFacility a rdfs:Class;
4   rdfs:comment "A parking facility containing parking spaces for vehicles
        (e.g car, motocycle)"@en .
5
6 on:ParkingManager a rdfs:Class;
7   rdfs:subClassOf gr:BusinessEntity.
8
9 mobivoc:entrance a rdf:Property;
10  rdfs:domain on:ParkingFacility;
11  rdfs:range mobivoc:CivicStructure;
12  rdfs:subPropertyOf dc:hasPart .
```

Listing 2.2: RDFS ontology example

Globally-scoped Identifier As mentioned above, RDFS only extends the RDF vocabulary and thus RDFS ontologies are directly written in RDF. Thus, RDFS contribute to this requirement as RDF already enforces the use of IRIs that are globally scoped. However, RDFS does not solve all ambiguities that may raised with regards to identifiers. When independent parties describe resources, they may use different IRIs that may refer to the same resource or different resources. Thus, even if IRIs are used in RDFS vocabularies, RDFS does not provide constructs for specifying whether two resources identified by different IRIs are the same or different.

Ontology Language RDFS contribute to this requirement by allowing the development of ontologies to fix the meaning of terms in two main ways. Firstly, a concise natural language definition can be specified for terms. For example, `on:ParkingFacility` is explicitly defined in natural language on Line 4 (Listing 2.2). Secondly, RDFS allows defining a structure among terms by setting inter-relationships between them. For example, the RDFS term `rdfs:subClassOf` (Listing 2.2, Line 9) is used to indicate a subclass relationship between `on:ParkingManager` and `gr:BusinessEntity`. However, RDFS is a basic ontology language and may not be enough expressive for domains where complex knowledge need to be represented. Our ontology in Listing 2.2 is simple but there are facts such as a disjointment between `on:ParkingFacility` and `on:ParkingManager` that cannot be expressed.

Semantics in Syntax As mentioned before, RDFS ontologies are written in RDF. Due to the flexibility of RDF described in Section 2.1.1, RDFS ontologies and the RDF data they describe can be merged to obtain a final RDF graph containing data together with its semantics. For example, the RDF graph in Listing 2.1 partly uses the ontology in Listing 2.2 that is also an RDF graph and both can be merged together to obtain a final graph. Thus, RDFS contributes to this requirement by allowing

data semantics to be shipped together with the data making it both self-described and portable.

While RDFS does contribute to the above requirements, its expressivity may not be enough for complex domains requiring more expressive ontology languages such as OWL that is discussed in the next section.

2.2.2 OWL

OWL extends RDFS to allow more expressive formalizations between classes and properties such as intersection of classes, cardinality restrictions on properties, etc. Thus, it reuses terms from RDFS but also introduces several new ones to represent the latter formalizations. OWL ontologies can be modularized that brings in the benefits of modularization such as separation of concerns, reusability, etc. Also, [CGHP+12] specifies a mapping between concepts of OWL and RDF meaning that OWL ontologies can be written as RDF graphs and serialized in RDF syntaxes.

The ontology in Listing 2.3 is an OWL ontology serialized in the Turtle syntax that is, as mentioned before, one of RDF syntaxes. Only part of the ontology is given and its remaining part is in a different module that is imported using the `owl:imports` predicate. Its informal presentation in UML is shown in Figure 2.2. It extends the RDFS ontology given in Listing 2.2 and therefore uses several terms from RDFS but also uses terms from OWL. It uses `owl:disjointWith` to state that `on:ParkingManager` and `on:ParkingFacility` are disjoint (Listing 2.3, Line 10). It also uses the object property `mobivoc:entrance` to link instances of `on:ParkingFacility` and `mobivoc:CivicStructure` (Listing 2.3, Line 12).

```
1 <http://example.com/ontology> a owl:Ontology;
2   owl:imports <http://example.com/ontology_part>
3   rdfs:label "An ontology for describing parking facilities"@en .
4
5 on:ParkingFacility a rdfs:Class, owl:Class;
6   rdfs:comment "A parking facility containing parking spaces for vehicles
        (e.g car, motocycle)"@en .
7
8 on:ParkingManager a rdfs:Class, owl:Class;
9   rdfs:subClassOf gr:BusinessEntity;
10  owl:disjointWith on:ParkingFacility .
11
12 mobivoc:entrance a rdf:Property, owl:ObjectProperty;
13  rdfs:domain on:ParkingFacility;
14  rdfs:range mobivoc:CivicStructure;
15  rdfs:subPropertyOf dc:hasPart .
```

Listing 2.3: OWL Ontology Example

Let us now further describe RDFS with respect requirements of data semantics described in Chapter 1 (Section 1.3.3).

Globally-scoped Identifier Like RDF, OWL also enforce the use of IRIs to name entities. OWL further contribute to this requirement by providing constructs to resolve the ambiguities related to IRIs that may be raised in RDFS. More precisely, it provides `owl:sameAs` to indicate that two IRIs refer to the same thing and `owl:differentFrom` to indicate that two IRIs refer to different things. For example,

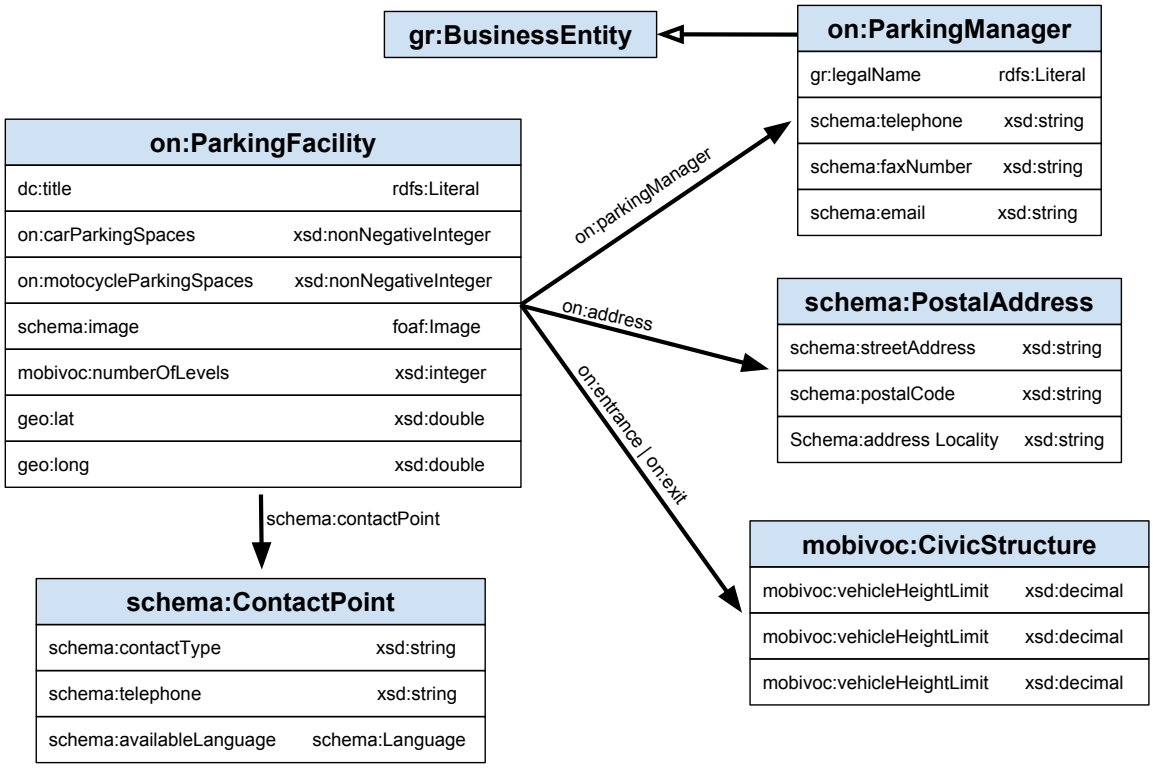

FIGURE 2.2: Informal presentation of Parking Ontology in UML

in Listing 2.1, we use `owl:sameAs` on Line 30 to explicitly state that `lang:fr` and `dbpedia:French_language` refer to the french language.

Ontology Language Unlike RDFS that is an extension of the RDF vocabulary, OWL is a language specifically designed to encode rich and complex domain knowledge. While OWL reuses terms from RDFS, it contributes further to this requirement by providing more constructs for defining relationships between terms that is important for different domains. In Section 2.2.1, we mentioned that RDFS cannot be used to describe disjointment. As we see in Listing 2.3 on Line 10, this is possible in OWL using `owl:disjointWith`.

Semantics in Syntax OWL is based on description logic and can be serialized in several syntaxes [MPSP12a, MPSP12b]. [CGHP+12] is a standard specification from the W3C that describes the serialization of OWL ontologies in RDF. For example, the RDF graph in Listing 2.1 uses the OWL ontology in Listing 2.3 that has been serialized as an RDF graph using the latter standard. Both RDF graphs can be merged to integrate the semantics with the data. Thus, like RDFS , OWL contributes to this requirement by making it possible for data semantics to be merged with the data making it both self-described and portable.

Both RDFS and OWL contributes to the requirements for data semantics. Let us consider tools to help in managing ontologies written using them.

2.2.3 RDFS/OWL Ontology Management

The generally accepted best practice before creating ontologies is to reuse existing ones wherever possible as doing so enhances semantic interoperability in an information ecosystem [KTM17]. To enhance the ontology reuse, search engines such as Swoogle [DFJ+04] or Watson [dM11] help users to find ontologies.

In case reusable ontologies are not found, they may be developed. The development process may be facilitated by ontology editors such as Web-Protégé [TVN08] or TopBraid Ontology Editor[1]. Also, the ontology development process may include a number of activities. Ontology engineering methodologies such as Methontology [FJ97] or DILIGENT [PST04] define the different activities to be carried out in the development process. Also, there are generators (e.g. FRED [GPR+17]) that may be used to generate ontologies from structured data or natural language description.

With RDF, we can encode data in the form triples and using RDFS and OWL, we have a way to add semantics to data. Let us describe in the next section how to provide access to this data.

2.3 Data Access

In this section, we describe the query language SPARQL [Gro13] and linked data platforms and explain how they can be used to enhance data interoperability at the data access level in highly decentralized information ecosystem. In the remainder of this section, we describe SPARQL and linked data platforms in Section 2.3.1 and Section 2.3.2.

2.3.1 SPARQL

SPARQL provides a set of standards that include languages and protocols to query and manipulate data in RDF. In this section, we provide an overview of the SPARQL query language in Section 2.3.1 and further describe it with respect to data access requirements identified in Chapter 1 (Section 1.3.3). Finally, in Section 2.3.1, we briefly describe some tools related to it.

Overview of SPARQL

The SPARQL query language is the standard language for querying RDF data. It is the SQL for the RDF data model. The core of a SPARQL query is the `WHERE` clause containing a *query pattern* that we formally describe in Chapter 4 (Section 4.2.1). An example of a SPARQL query is given in Listing 2.4 with the query pattern being `{...}` (Line 2 - 8).

In short, a query pattern defines variables and constraints they should satisfy with respect to an RDF graph. Triple patterns are the simplest way to define these constraints. A triple pattern is like an RDF triple except that subject, predicate and object can be a variable. In the query pattern given in Listing 2.4, `?parkingFacility a on:ParkingFacility` (Line 3) is a triple pattern and `?catalog` a variable. Variables

[1] `https://www.topquadrant.com/2013/06/10/the-topbraid-evn-ontology-editor/`, last accessed 16 September 2018

may be further constrained using filter expression. The filter expression `(?p != on:parkingManager)` (Line 7) states that the solutions for `?p` should not include `on:parkingManager`.

```
1 SELECT ?parkingFacility WHERE
2 {
3   ?parkingFacility a on:ParkingFacility;
4      ?p ?o;
5      schema:contactPoint ?contactPoint .
6   ?contactPoint ?p1 ?o1 .
7   FILTER (?p != on:parkingManager)
8 }
```

Listing 2.4: SPARQL SELECT query example

Evaluating a query pattern with respect to an RDF graph returns a set of solution mappings where in every solution, terms from the RDF graph are mapped to a variable. For example, executing the above query pattern with respect to the RDF graph in Listing 2.1 may return the set of solution mappings in Table 2.1 with every row being a solution.

?pFacility	?p	?o	?p1	?o1	?contactPoint
ex:Anvers4	rdf:type	on:ParkingFacility	schema:telephone	"0754234448"	ex:EmergencyCP
ex:Anvers4	dc:title	"Anvers Parking"	schema:availableLanguage	lang:fr	ex:EmergencyCP
ex:Anvers4	dc:title	"Anvers Parking"	schema:contactType	"Emergency"	ex:EmergencyCP

Table 2.1: Set of solutions mappings

The SPARQL query may also contain *solution modifiers* that further process results obtained from the evaluation of a query pattern. `LIMIT` is an example of a solution modifier and is parameterized by a non-negative n denoting the maximum number of results to return. Applying `LIMIT` with $n = 1$ on the set of the solution mappings in Table 2.1 may return only the first row.

Finally, a SPARQL query has a query form that generates the final result from a set of solution mappings obtained from the evaluation of the query pattern and solution modifiers (if any). There are four query forms namely `SELECT`, `CONSTRUCT`, `ASK` and `DESCRIBE`. In the paragraphs below, we describe these query forms using examples of SPARQL queries having same query pattern in their `WHERE` clause like the query in Listing 2.4. Also, we directly use the set of solution mappings in Table 2.1 as the result of this query pattern.

A SPARQL `SELECT` query takes a set of variables and returns their bindings for each solution mapping. The query in Listing 2.4 is in fact a SPARQL `SELECT` query. With respect to the set of solution mappings in Table 2.1, this query return only the first column as its set of variables contains only `?pFacility`.

A SPARQL `CONSTRUCT` query has a set of triple patterns using which RDF graphs are created from a set of solution mappings. For example, using the CONSTRUCT query in Listing 2.5 with respect to to the set of solution mappings in Table 2.1 returns the new RDF graph in Listing 2.6. This RDF graph is obtained by using every triple pattern to generate triples by replacing its variables with their bindings from every solution mapping. Consider the example of the first triple pattern `?pFacility ?p ?o` from Listing 2.5. Using their bindings from the first solution mapping in Table 2.1, the first triple in the RDF graph in Listing 2.6 is generated. Using the same triple pattern and the second solution mapping in Table 2.1, the fourth triple from

the same RDF graph is generated. A solution mapping is ignored if it does not have bindings for all the variables in the triple pattern.

```
1  CONSTRUCT {
2    ?pFacility ?p ?o .
3    ?contactPoint ?p1 ?o1 .
4  } WHERE {
5    ?pFacility a on:ParkingFacility;
6      ?p ?o;
7      schema:contactPoint ?contactPoint .
8    ?contactPoint ?p1 ?o1 .
9    FILTER (?p != on:parkingManager)
10 }
```

Listing 2.5: CONSTRUCT query example

```
1  ex:Anvers4 a on:ParkingFacility ;
2    on:carParkingSpaces 100 ;
3    on:motocycleParkingSpaces 50 ;
4    dc:title "Anvers" ;
5    schema:contactPoint ex:EmergencyCP ;
6    geo:lat 48.83523833495664 ;
7    geo:long 2.37322158810141 .
8
9  ex:EmergencyCP schema:
       availableLanguage lang:en ,
10                   lang:fr;
11     schema:contactType "Emergency
           Contact Point" ;
12     schema:telephone "0754234448" .
```

Listing 2.6: Result of CONSTRUCT query

SPARQL `ASK` queries returns boolean answers to demonstrate existence or inexistence of solutions. For example, the `ASK` query in Listing 2.7 return $true$ with respect to the set of solution mappings in Table 2.1 .

```
1 ASK WHERE {
2   ?parkingFacility a on:ParkingFacility;
3   ?p ?o;
4   schema:contactPoint ?contactPoint .
5   ?contactPoint ?p1 ?o1 .
6   FILTER (?p != on:parkingManager)
7 }
```

Listing 2.7: SPARQL SELECT query example

Finally, SPARQL `DESCRIBE` queries returns descriptions of resources but its semantics is determined by the implementation of SPARQL engines [HS13b, §16.4.3]. For example, the `DESCRIBE` query in Listing 2.8 may return only the portion of the RDF graph in Listing 2.6 where `ex:Anvers4` is the subject.

```
1 DESCRIBE ex:Anvers4 WHERE {
2   ?pFacility a on:ParkingFacility;
3         ?p ?o;
4   schema:contactPoint ?contactPoint .
5   ?contactPoint ?p1 ?o1 .
6   FILTER (?p != on:parkingManager)
7 }
```

Listing 2.8: SPARQL SELECT query example

Let us now further describe SPARQL with respect to data access requirements identified in Chapter 1 (Section 1.3.3).

Standardized High-Level Data Access As mentioned before, the SPARQL query language is itself standardized. In addition to this, the Graph Store HTTP Protocol [Ogb13] is another standard from W3C that allows communicating with SPARQL engines via HTTP. Using these two standards, high-level data access systems commonly known as SPARQL endpoints can be setup enabling data access without having to bother about implementation-specific details of servers and clients. These two standards contribute to this requirement by enabling the setup of high-level data access systems commonly known as SPARQL endpoints to allow data exchanges between servers and clients without having to bother about implementation-specific details.

High-Level Data Access Description The SPARQL query language has a formal syntax and semantics that describe the validity of queries and their excepted evaluation results. Similarly, the Graph Store HTTP Protocol describes the HTTP requests and responses by specifying details such as HTTP headers, payload, status codes, etc. In short, these two standards contributes to this requirement by providing sufficient information for formulating information requests as SPARQL queries, for sending them in HTTP requests to SPARQL endpoints and for interpreting the responses.

As we have seen, SPARQL contributes to both requirements and can be used for high-level data access. Let us now consider tools related to SPARQL in the next section.

Tools Related to SPARQL

Systems that uses RDF natively store their data in repositories commonly known as triple stores. The triple stores is normally made up of several software that implement SPARQL standards for manipulating the RDF data. There exist a number of triplestores [NS14, PZLN18] such as *Apache Jena TDB*[1], *Virtuoso*[2], etc.

The most common software bundled together with triple stores are SPARQL engines and SPARQL servers. A SPARQL engine process SPARQL queries and returns results based on the SPARQL query results standards [Sea13b, Haw13, Sea13a]. Some free and open source SPARQL engines are Apache Jena ARQ [3] and RDF4J [4]. SPARQL servers provides access to SPARQL engines via HTTP. An example is Apache Fuseki[5] that uses Apache Jena ARQ as its SPARQL engine and Apache Jena TDB and its triple store.

Under high loads, SPARQL servers may be unavailable. There are number of tools to handle this issue. Linked Data Fragments is a technology that address this issue by redistributing the load between delegating part of the processing to clients [VHM+14]. Triple pattern fragments is a specific type of Linked Data Fragments that provides an interface to triples which client-side applications may access exploit to evaluate SPARQL queries thus decreasing load on the server [VSH+16]. ULYSSES [MSMV18] is such a client-side application that further take advantage of replications to decrease the load on triple pattern fragments servers.

Besides, there are other tools such as query editors (e.g. NITELIGHT [RS08] and YASGUI [RH13]) that facilitate query writing or query generators (e.g. hsparql[6], spanqit[7]) to programmatically create and generate queries.

SPARQL for High-Level Data Access

As we have seen in Section 2.3.1, SPARQL satisfies the requirements for enhancing interoperability at data access level. However, using SPARQL as a high-level data access systems poses two problems.

[1] https://jena.apache.org/documentation/tdb/, last accessed 27 July 2018
[2] https://virtuoso.openlinksw.com/, last accessed 27 July 2018
[3] https://jena.apache.org/documentation/query/, last accessed on 16 April 2018
[4] http://rdf4j.org/, last accessed on 16 April 2018
[5] https://jena.apache.org/documentation/fuseki2/,last accessed on 16 April 2018
[6] http://hackage.haskell.org/package/hsparql, last accessed 30 July 2018
[7] https://github.com/anqit/spanqit, last accessed 30 July 2018

The first problem is that data consumers need to have knowledge of the data. More precisely, to formulate information requests, they need to know the different ontologies per which the data is structured as well as naming of resources in the data. Moreover, to formally express these information requests as SPARQL queries, they need to know the syntax and semantics of the SPARQL query language.

The second problem is that data publishers may require much resources in their environment to host a high-level data access system based on SPARQL as SPARQL queries may have a high computational complexity [PAG09]. While optimizations are possible, doing so may be complex as it requires making assumptions about the data consumers such as types of queries they may ask or their ability to participate in query evaluation using technologies such as Linked Data Fragments that we have discussed in Section 2.3.1.

Besides these two problems, let us make an analogy using XQuery and SQL that are query languages for XML and relational databases. On the Web, there are a number of SPARQL endpoints. [AHUV13] makes a survey of 427 SPARQL endpoints. However, to our knowledge, there is hardly any system providing access to their data direct using XQuery or SQL. We believe the reason for this to be that these technologies are best for a system's internal use and in a system architectured using the model-view-controller pattern, we would expect these technologies to be at the controller level hidden to data consumers by the view. Normally, Web APIs are usually used at the view level to provide a high-level data access to end-users. For example, ProgrammableWeb[1] documents over nineteen thousands APIs that provide access to different types of data on the Web.

Thus, we believe that while SPARQL can be used at the controller level, there is a need for a technology at the view layer to hide the use of SPARQL. Let us now analyze in the next section whether platforms based on linked data can be used at the view layer.

2.3.2 Linked Data-based Platform

Linked Data-based platforms provide RESTful access to RDF description of resources. In this section, in Section 2.3.2, we provide an overview of such platforms and in Section 2.3.2, we describe tools that facilitate their instantiation.

Overview of Linked Data-Based Platform

Linked Data-based platforms are based on the four Linked Data principles [BL06]:

- Use IRIs to name (identify) things;
- Use HTTP IRIs so that these things can be looked up (dereferenced);
- Provide useful information when dereferencing things using open standards (e.g. RDF, SPARQL)
- Include links to other things using HTTP IRIs to enhance knowledge discovery.

[1] https://www.programmableweb.com/apis, last accessed 18 September 2018

Let us consider a platform based on the above principles. Suppose we want to expose the resource `ex:Anvers4` described in Listing 2.1 via a Linked Data-based platform at its IRI itself. Dereferencing `ex:Anvers4` returns its description in Listing 2.6 that satisfies the four Linked Data principles. The first and second principles are satisfied by using HTTP IRIs. The third principles is satisfied because dereferencing `ex:Anvers4` returns its description in RDF. Finally, the fourth principle is satisfied since the `ex:Anvers4`'s description include the externally described resource `lang:fr` (Listing 2.6, Line 10) which as per its publisher's guidelines[1] can be dereferenced with the MIME type `application/rdf+xml` to obtain its RDF description in `RDF/XML` [CS14a] syntax. As we can see, by following the above principles, Linked Data-based platforms behave like Web APIs and provide access to resource description without data consumers having to know about any technicalities.

Let us now further describe Linked Data-based platforms with respect to data access requirements identified in Chapter 1 (Section 1.3.3).

Standardized High-Level Data Access Linked Data principles brings together concepts from HTTP, RDF and several other standards. However, it leaves many open choices such as default RDF syntax to use, headers in HTTP requests when creating Linked Data resources or HTTP methods to use for different operations on Linked Data resources. Thus, while Linked Data principles does contribute to this requirement by using existing standards, platforms implemented based on it with the open choices being characterized by independent parties may generate data access heterogeneity.

High-Level Data Access Description In the case of SPARQL, as described in Section 2.3.1, both its query language and the Graph Store Protocol describe the validity of HTTP requests to SPARQL endpoints and also interpretation of responses. However, Linked data principles does not specify any such description and thus leaves room for much interpretation. While documentations may be provided by data publishers to describe Linked Data-based platforms, a documentation for every platform enhances tight coupling making both data exploitation and development of generic tools more complex. Thus, Linked Data-based platform may provide documentation to help data consumers to exploit it, the Linked Data principles in itself does not satisfy this requirement.

Let us consider tools that facilitate the implementations of Linked Data principles.

Linked Data-based Platforms Implementation

There are tools that automate the generation of linked data-based platforms by implementing Linked Data principles [BHB09] and they can be categorized based on the type of data source from which they generate the platforms.

Firstly, Pubby[2] consider only RDF data sources. Using a mapping, it can generate IRIs for linked data resources from resources in the RDF data source that can be SPARQL endpoints or static RDF files. When requests are made on linked data

[1] `https://id.loc.gov/techcenter/serializations.html`, last accessed on 30 July 2018
[2] `http://wifo5-03.informatik.uni-mannheim.de/pubby/`, last accessed on 18 September 2018

resources, their corresponding RDF resources are obtained and finally SPARQL DESCRIBE queries are used to return their descriptions.

Then, D2R Server [BC06] and Triplify [ADL+09] can consider relational databases. D2R Server [BC06] can provide linked data using RDF graphs generated from relational data through mappings written in the D2RQ mapping language [BS04]. Triplify [ADL+09] maps HTTP-URI requests to SQL queries whose results are then transformed into RDF from linked data resources are generated. To facilitate its usage, it provides a library of mappings for popular web applications such as WordPress[1] and Drupal[2].

Finally, there is Virtuoso Universal Server[3], SparqPlug [CHM08] and RDF-REST [Cha13a]. Virtuoso Universal Server can consider both RDF sources and relational databases and for the latter, it uses their mappings to RDF graphs. SparqPlug can consider HTML documents as data sources. It does so by serializing the HTML DOM as RDF on which it evaluates user-defined SPARQL queries to generate RDF for linked data resources. RDF-REST is a framework that implement REST and Linked Data principles. Its aim is to facilitate the development of RESTful web services that can produce linked data from other linked data sources or heterogeneous web APIs.

In spite of the above tools, as mentioned in Section 2.3.2, Linked Data-based platforms can generate data access heterogeneity due to the open choices that need to be characterized during their set up. Therefore, to limit this heterogeneity, there is a need for a standard that characterize the core aspects of Linked Data-based platforms. This was the main motivation of the *Linked Data Platform Working Group*[4] [ABH15] that resulted into the *Linked Data Platform 1.0 W3C Recommendation* that we describe in the next section.

2.4 Linked Data Platform 1.0

Linked Data Platform 1.0 is a W3C recommendation aims to standardize techniques for working with Linked Data over HTTP. It extends the Linked Data principles to provide an architecture for read-write Linked Data. We refer to Linked Data Platforms (LDP) as platforms implementing the latter standard and *LDP standard* as the standard itself. We reserve a complete section complete for LDPs and describe it in more detail as it is the core of this thesis.

In the remainder of this section, we provide an overview of the standard in Section 2.4.1. Then, in Section 2.4.2, we describe an illustrating example that reuses the examples from the previous sections. We describe existing LDP implementations in Section 2.4.3. Finally, in Section 3.1.1, we describe the development of LDPs for exposing data from existing data sources and the problems that may occur while doing so.

[1] https://wordpress.com/, last accessed on 9 April 2018
[2] https://www.drupal.org/, last accessed on 9 April 2018
[3] https://virtuoso.openlinksw.com/, last accessed 30 July 2018
[4] https://www.w3.org/2012/ldp/wiki/Main_Page, last accessed 4 August 2018

2.4.1 Overview of LDP Standard

The LDP standard consists of two parts, the domain model and the interaction model. The domain model describes the organization of LDP resources that consist of different types of resources as shown in Figure 2.3. The interaction model describes interactions for to perform read-write operations on these resources via HTTP methods. For example, it describes requests and replies details (e.g. headers, payload) of several HTTP methods out of which `GET` and `OPTIONS` must be supported while the remaining ones are optional.

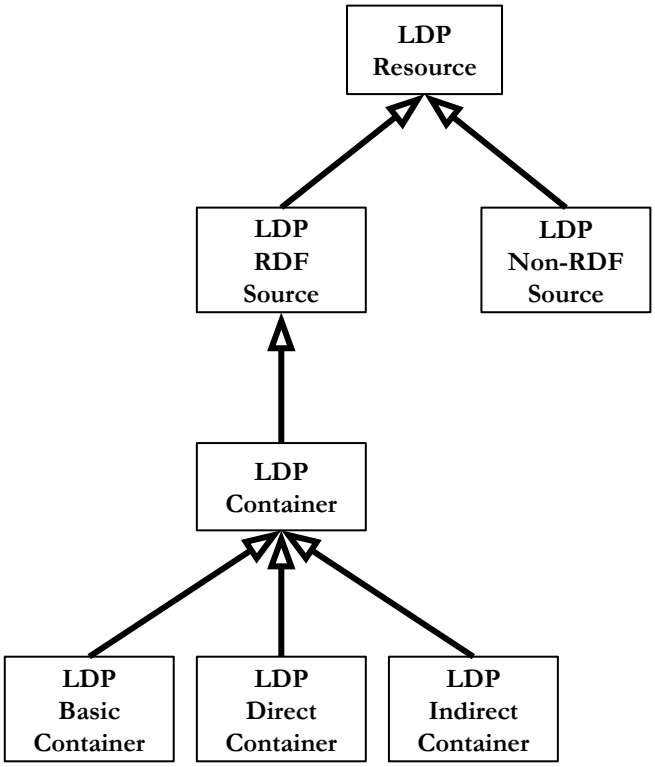

Figure 2.3: Overview of LDP Domain Model

The domain model categorizes LDP resources into LDP RDF Sources (LDP-RS) and LDP Non-RDF Sources (LDP-NR). The state of an LDP-RS is represented in RDF contrary to that of an LDP-NR whose state is not in RDF. LDP containers are specializations of LDP-RSs that organizes resources, also known as their members, and manage requests from clients for their creation, modification, deletion or enumeration based on their interaction model. There are three types of types LDP containers namely LDP Basic Container (LDP-BC), LDP Direct Container (LDP-DC) and LDP Indirect Container (LDP-IC) that mostly vary on the expressivity they provide to describe their members. We provide a detailed description of them in the next section using a concrete example.

2.4.2 Detailed Description

In this section, we provide a detailed description of the LDP standard with a focus on LDP containers. In the remainder of this section, we reuse code snippets that we

have been using since the beginning of this chapter to describe basic, direct and indirect containers in Section 2.4.2, Section 2.4.2 and Section 2.4.2 respectively.

Basic Containers

Basic containers defines a simple containment relationship with its resources using the predicate `ldp:contains` from the LDP vocabulary[1]. The set of triples in its representation that uses this property is called *containment triples* and in every of them, the IRI of the subject and object is that of the basic container and the corresponding LDP resource respectively.

Let us illustrate the use of basic containers using an example. We want to create a basic container to describe the parking facility in Listing 2.6 and then as we proceed the example, we are going add other resources in it. Assuming the existence of a basic container `dex:ParkingFacilities` storing parking facilities, to create a container in it, the `POST` request in Listing 2.9 is sent to it. Some aspects from the `POST` requests that are characterized by the LDP interaction model are:

- the null relative IRI (<>) in message body of the `POST` request is used as a placeholder for the IRI of the LDP resource to be created [SAM15c, §5.2.3.7];
- the `Link` header informs the server about the type of LDP resource to be created and in our case, it is a basic container [SAM15c, §5.2.3.4];
- the `Slug` header, originally defined in [Gdh07, §9.7], is used by the LDP standard to provide suggest a string that may used be in the server's final choice of resource IRI [SAM15c, §5.2.3.10], and in our case this string is `Anvers4`;

```
1  Request URL: http://data.example.com/ParkingFacilities
2  Method: POST
3  Content-Type: text/turtle
4  Link: <http://www.w3.org/ns/ldp#BasicContainer>; rel="type"
5  Slug: Anvers4
6  Message Body:
7  <> a ldp:BasicContainer .
8  <#it> a on:ParkingFacility ;
9    on:carParkingSpaces 100 ;
10   on:motocycleParkingSpaces 50 ;
11   dc:title "Anvers" ;
12   geo:lat 48.835 ;
13   geo:long 2.3732;
14   foaf:primaryTopic ex:Anvers4 .
```

Listing 2.9: POST Request for creating basic container

In the message body of the `POST` request, we use the resource <#> instead of using `ex:Anvers4` directly. We explain the reason for doing so in the next section. Suppose that the above `POST` request is successfully processed, the server may return the response in Listing 2.10.

```
1  Status Code: 201 Created
2  Location: http://data.example.com/Anvers4
3  Link: <http://www.w3.org/ns/ldp#BasicContainer>; rel="type",
4      <http://www.w3.org/ns/ldp#Resource>; rel="type"
```

Listing 2.10: POST response from creating the basic container

[1] https://www.w3.org/ns/ldp, last accessed 1 August 2018

The response provides the client with the following pieces of information required by the LDP interaction model:

- the status code `201` informs that the `POST` request resulted in the successful creation of the resource [SAM15c, §5.2.3.1];
- the `Location` header provides the final IRI of the created resource [SAM15c, §5.2.3.1], and in our case it is `dex:Anvers4`;
- the `Link` header in `POST` responses informs about LDP interactions can be performed on the new resource, and in our it is LDP interactions that relates to LDP resources and basic containers.

Now, a simple GET request on the resource `dex:ParkingFacilities` returns the response in Listing 2.11 with its representation in the message body. As we can see, in the representation, a containment triple (Line 10) is added to the container `dex:ParkingFacilities` to specify its new member `dex:Anvers4`. Besides this, the LDP standard uses several headers such as `ETag`, `Allow` or `Accept-Post` to convey other information that is not of particular use in our example.

```
1  Status Code: 200 OK
2  Content-Type: text/turtle
3  ETag: "87e52ce291112"
4  Link: <http://www.w3.org/ns/ldp#BasicContainer>; rel="type",
5        <http://www.w3.org/ns/ldp#Resource>; rel="type"
6  Accept-Post: text/turtle, application/ld+json
7  Allow: POST,GET,OPTIONS,HEAD
8  Message Body:
9  dex:ParkingFacilities a ldp:BasicContainer;
10         ldp:contains dex:Anvers4 . # containment triple
```

Listing 2.11: GET Response on `dex:ParkingFacilities` after addition of dex:Anvers4

With the above example, we provided a brief idea about basic containers. As we have seen, basic containers use containment triples to state their members and in our examples, the fact that these triples cannot be changed does not create a problem. However, suppose that we want to add images for the new parking facility and we want to continue using the predicate `schema:image`, per our ontology in Figure 2.2, to link the parking facility defined in `dex:Anvers4` to the image. Doing so is not possible because as we mentioned, containment triples cannot be altered to contain client-specific resources. Let us now see how direct containers can be used in this situation. Let us now see how direct containers can be used in this situation.

Direct Container

Direct containers extends basic containers by introducing membership triples that provides more flexibility than containment triples. They enable the use of predicates from domain vocabularies and allow either the subject or object of membership triples to refer to resources that may not be LDP resources (e.g. real-world entities).

To show the flexibility of direct containers, we continue the example of adding images for the parking facility in `dex:Anvers4`. Two design choices may be considered when doing so:

- new images can be created directly in `dex:Anvers4`;

- a specific container can be created in dex:Anvers4 to store images.

The advantage of the second design choice is that it allows referring to a set of images directly. Assuming that the second design choice is used, we create the direct container for storing images by sending the POST request in Listing 2.12 to the basic container dex:Anvers4. Notice the ldp:membershipResource and ldp:hasMemberRelation predicates in the message body of the POST request that are used to specify the subject and predicate of membership triples respectively. We will see their usage when adding images below.

```
1 Request URL: http://data.example.com/Anvers4
2 Method: POST
3 Content-Type: text/turtle
4 Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type"
5 Slug: Anvers4Images
6 Message Body:
7 <> a ldp:DirectContainer;
8  ldp:membershipResource http://data.example.com/Anvers4#it;
9  ldp:hasMemberRelation schema:image .
```

Listing 2.12: POST request for creating a direct container in dex:Anvers4

Suppose that the POST request in Listing 2.12 is successfully processed with the new direct container created with the IRI dex:Anvers4Images. After this, a simple GET request on dex:Anvers4 returns the response in Listing 2.13 with its representation in the message body. As we see, a containment triple (Line 10) is added to the container dex:Anvers4 for the new container dex:Anvers4Images.

```
1 Status Code: 200 OK
2 Content-Type: text/turtle
3 ETag: "90e52ce291112"
4 Link: <http://www.w3.org/ns/ldp#BasicContainer>; rel="type",
5      <http://www.w3.org/ns/ldp#Resource>; rel="type"
6 Accept-Post: text/turtle, application/ld+json
7 Allow: POST,GET,OPTIONS,HEAD
8 Message Body:
9 dex:Anvers4 a ldp:BasicContainer;
10  ldp:contains dex:Anvers4Images . #Containment triple
11
12 <#it> a on:ParkingFacility ;
13  on:carParkingSpaces 100 ;
14  on:motocycleParkingSpaces 50 ;
15  dc:title "Anvers" ;
16  geo:lat 48.835 ;
17  geo:long 2.3732;
18  foaf:primaryTopic ex:Anvers4 .
```

Listing 2.13: GET response on ex:Anvers4 after adding dex:Anvers4Images

Also, a GET request on the new container dex:Anvers4Images returns the response in Listing 2.14. As we can see, in the message body of the response, container does not contain any members so far.

```
1 Status Code: 200 OK
2 Content-Type: text/turtle
3 ETag: "91e52ce291112"
4 Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type",
5 <http://www.w3.org/ns/ldp#Resource>; rel="type"
6 Accept-Post: text/turtle, application/ld+json
7 Allow: POST,GET,OPTIONS,HEAD
8 Message Body:
9  dex:Anvers4Images a ldp:DirectContainer;
10  ldp:membershipResource http://data.example.com/Anvers4#it;
11  ldp:hasMemberRelation schema:image .
```

Listing 2.14: Representation of Direct container ex:Anvers4Images

Now that the container dex:Anvers4Images for holding images has been created, let us add an image to it by sending the POST request in Listing 2.15 with the encoded image file.

```
1 Request URL: http://data.example.com/Anvers4Images
2 Method: POST
3 Content-Type: image/png
4 Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
5 Slug: Anvers4Image1
6 Message Body:
7 ### binary data for image ####
```

Listing 2.15: POST request for creating image in dex:Anvers4Images

Assuming the POST request in Listing 2.15 is successfully processed with the IRI dex:Anvers4Image1 assigned to the newly non-RDF resource (i.e. the image), at least the following two new triples will be created:

- the containment triple (Line 12), as we can see in the GET response in Listing 2.16, is added to dex:Anvers4Images for the new member, the image dex:Anvers4Image1;
- the membership triple (Line 19), as we can see in the GET response in Listing 2.17, is added to dex:Anvers4 with its subject and object generated considering the ldp:membershipResource and ldp:hasMemberRelation defined in direct container (Listing 2.12).

```
1  Status Code: 200 OK
2  Content-Type: text/turtle
3  ETag: "91f52ce291112"
4  Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type",
5        <http://www.w3.org/ns/ldp#Resource>; rel="type"
6  Accept-Post: text/turtle, application/ld+json
7  Allow: POST,GET,OPTIONS,HEAD
8  Message Body:
9    dex:Anvers4Images a ldp:DirectContainer;
10     ldp:membershipResource http://data.example.com/Anvers4#it;
11     ldp:hasMemberRelation schema:image;
12     ldp:contains dex:Anvers4Image1 . # containment triple
```

Listing 2.16: GET response on dex:Anvers4Images after adding image

```
1  Status Code: 200 OK
2  Content-Type: text/turtle
3  ETag: "91g52ce291112"
4  Link: <http://www.w3.org/ns/ldp#BasicContainer>; rel="type",
5  <http://www.w3.org/ns/ldp#Resource>; rel="type"
6  Accept-Post: text/turtle, application/ld+json
7  Allow: POST,GET,OPTIONS,HEAD
8  Message Body:
9  <> a ldp:BasicContainer;
10   ldp:contains dex:Anvers4Images .
11
12 <#it> a on:ParkingFacility .
13   on:carParkingSpaces 100 ;
14   on:motocycleParkingSpaces 50 ;
15   dc:title "Anvers" ;
```

```
16   geo:lat 48.835 ;
17   geo:long 2.3732;
18   schema:image dex:Anvers4Image1; #membership triple
19   foaf:primaryTopic ex:Anvers4 .
```

Listing 2.17: GET response on dex:Anvers4 after adding image

In the above example, we were able to materialize the membership triples in dex:Anvers4 because we used the resource <#it>[1] as the membership resource in the direct container dex:Anvers4Images instead of ex:Anvers4. We intuitively did so following the examples in the LDP standard, though non-normative, that materialize membership triples in the membership resource. However, the LDP standard is not clear on this issue. In fact, we believe that the LDP standard has some ambiguities regarding membership resources and membership triples. First, it does not explicitly specify whether the membership resource should be an LDP resource on the current server where the direct container is. Moreover, as we mentioned, it does not explicitly specify the location where membership triples should be materialized.

In short, compared to basic containers, direct containers introduce the notion of membership triples whose membership resource and predicate can be specified by the user. However, the member resource must be an LDP resource and consequently its IRI is determined by the server [SAM15c, §5.4.1.5]. For example, in the membership tripe in dex:Anvers4, the object is dex:Anvers4Image1 and cannot be changed. This is where the limitation of direct container lies. In our example, this was not a problem. However, suppose that we want to add parking managers to the parking facility in dex:Anvers4 using their original IRIs, this is not possible using direct containers. Let us illustrate in the next section how using indirect containers allow us to do so.

Indirect Containers

To store parking managers for the parking facility in dex:Anvers4, we create an indirect container by sending the POST request in Listing 2.18 to dex:Anvers4.

```
1  Request URL: http://data.example.com/Anvers4
2  Method: POST
3  Content-Type: text/turtle
4  Link: <http://www.w3.org/ns/ldp#IndirectContainer>; rel="type"
5  Slug: Anvers4PMs
6  Message Body:
7  <> a ldp:IndirectContainer;
8    ldp:membershipResource http://data.example.com/Anvers4#it;
9    ldp:hasMemberRelation on:parkingManager;
10   ldp:insertedContentRelation foaf:primaryTopic .
```

Listing 2.18: POST request for creating container in dex:Anvers4

Compared to the representation of direct container in Listing 2.12, notice the indirect container's property ldp:insertedContentRelation on Line 10 (Listing 2.18). Briefly, it allows retrieving a resource from the newly created LDP RDF source to be used in the membership triple. We will see its use when creating parking managers below.

[1]This resource is in dex:Anvers4 and its absolute IRI is http://data.example.com/Anvers4#it

Suppose that the indirect container is successfully created with IRI dex:Anvers4. After this, a GET request on dex:Anvers4PMs returns the response in Listing 2.19. As we can see in the message body, a containment triple (Line 11) is added to dex:Anvers4.

```
1  Status Code: 200 OK
2  Content-Type: text/turtle
3  ETag: "91h52ce291112"
4  Link: <http://www.w3.org/ns/ldp#BasicContainer>; rel="type",
5       <http://www.w3.org/ns/ldp#Resource>; rel="type"
6  Accept-Post: text/turtle, application/ld+json
7  Allow: POST,GET,OPTIONS,HEAD
8  Message Body:
9  dex:Anvers4 a ldp:RDFSource, ldp:BasicContainer;
10   ldp:contains dex:Anvers4Images;
11   ldp:contains dex:Anvers4PMs . #Membership Triple
12
13 <#it> a on:ParkingFacility;
14   on:carParkingSpaces 100 ;
15   on:motocycleParkingSpaces 50 ;
16   dc:title "Anvers" ;
17   geo:lat 48.835 ;
18   geo:long 2.3732;
19   schema:image dex:Anvers4Image1; #membership triple
20   foaf:primaryTopic ex:Anvers4 .
```

Listing 2.19: Indirect container for parking managers

Now that the container dex:Anvers4PMs for storing parking managers has been created, let us create a parking manager by sending the POST request in Listing 2.20 to it.

```
1  Request URL: http://data.example.com/Anvers4PMs
2  Method: POST
3  Content-Type: text/turtle
4  Slug: Anvers4PM1
5  Message Body:
6  <> a ldp:RDFSource;
7    foaf:primaryTopic ex:Anvers4PM1 .
8  ex:AnversPM1 a on:ParkingManager;
9    gr:legalName "Anvers Parking Manager";
10   schema:telephone "0754234448";
11   schema:email "parking@anvers.com" .
```

Listing 2.20: POST request for creating new parking manager

Assuming the request is successfully processed with the IRI dex:Anvers4PM1 assigned to the newly created resource, at least two new triples will be added. Firstly, as we can see in the GET response (Listing 2.21) on dex:Anvers4PMs, a containment triple (Line 13) is added for the new resource dex:Anvers4PM1.

```
1  Status Code: 200 OK
2  Content-Type: text/turtle
3  ETag: "95h52ce291112"
4  Link: <http://www.w3.org/ns/ldp#IndirectContainer>; rel="type",
5  <http://www.w3.org/ns/ldp#Resource>; rel="type"
6  Accept-Post: text/turtle, application/ld+json
7  Allow: POST,GET,OPTIONS,HEAD
8  Message Body:
9  <> a ldp:IndirectContainer;
10   ldp:membershipResource http://data.example.com/Anvers4#it;
11   ldp:hasMemberRelation on:parkingManager;
12   ldp:insertedContentRelation foaf:primaryTopic;
13   ldp:contains dex:Anvers4PM1 . #Containment triple
```

Listing 2.21: GET response after adding dex:Anvers4PM1

Secondly, as we can see in the GET response (Listing 2.22) on dex:Anvers4, a membership triple (Line 20) is added to dex:Anvers4 to link the parking facility with the new parking manager.

```
1  Status Code: 200 OK
2  Content-Type: text/turtle
3  ETag: "91i52ce291112"
4  Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type",
5  <http://www.w3.org/ns/ldp#Resource>; rel="type"
6  Accept-Post: text/turtle, application/ld+json
7  Allow: POST,GET,OPTIONS,HEAD
8  Message Body:
9  <> a ldp:BasicContainer;
10   ldp:contains dex:Anvers4Images;
11   ldp:contains dex:Anvers4PMs .
12
13 <#it> a on:ParkingFacility;
14   on:carParkingSpaces 100 ;
15   on:motocycleParkingSpaces 50 ;
16   dc:title "Anvers" ;
17   geo:lat 48.835 ;
18   geo:long 2.3732;
19   schema:image dex:Anvers4Image1;
20   on:parkingManager ex:Anvers4PM1; #membership triple
21   foaf:primaryTopic ex:Anvers4 .
```

Listing 2.22: GET response after adding dex:Anvers4PM1

Notice that the object of the membership triple is now a different resource from the newly generated LDP RDF source, i.e. dex:Anvers4PM1 and is generated using the property ldp:insertedContentRelation and its value foaf:primaryTopic of the indirect container. More precisely, it tells the LDP server that the resource to be used in the object position of the membership triple should is indicated by the property foaf:primaryTopic of the newly created LDP RDF source dex:Anvers4PM1.

To resume, direct containers only allow customizing the membership resource and predicate in membership triples. Indirect contains extends on this and the provide the flexibility to even specify members that may be both LDP resources and external resources.

Synthesis

As we have seen in the previous section, the LDP standard provides different types of LDP resources for different use cases and defines forms of HTTP requests and responses to interact with these resources. Let us now describe the LDP standard with respect to data access requirements identified in Chapter 1 (Section 1.3.3).

Standardized High-Level Data Access The LDP standard contribute to this requirement by providing a standard way to expose and access Linked Data resouces. It extends the Linked Data principles and enable both read and write operations on resources. Also, it describes part of the standard that must be implemented by all LDP servers. Consequently, this enhances interoperability between LDP servers and clients and enable development of generic LDP server and client applications.

High-Level Data Access Description The LDP standard contribute to this requirement by defining and describing the interaction model and domain model. The interaction model provide a detailed description of what constitute valid LDP

requests and responses and specify the use of HTTP primitives (e.g. headers, status codes) for different types of read and write operations. Also, the data model specify terms, together with their semantics,that may appear in the payload of LDP responses. The interpretation of these terms are described in an OWL ontology, the LDP Vocabulary[1].

2.4.3 LDP Related Work

As we have seen, the LDP standard satisfies the requirements for enhancing data access interoperability meaning that in highly decentralized information ecosystems, data publishers may expose data from data providers using LDPs. In this section, we describe to what extent current works related to LDP can help when generating LDPs from existing data sources. They can be categorized into the peer reviewed scientific work and LDP implementations.

[HNS12] is the core work that has given rise to the LDP standard. The remaining scientific literature having a focus on LDP is limited to only four works [MGG13, MPC+14, MGG14, LIG+16] to our knowledge. [MGG13] highlights the benefits of LDPs for enterprise application integration. [MPC+14] demonstrates the generation of LDPs from simple R2RML mappings (no SQL view, no multiple mappings to a class/property) using hardcoded transformations without any customizing the design of output LDP resources. [MGG14] demonstrates the use of a hardcoded LDP adapter targeted only to the domain model of Bugzilla bug tracker[2]. Finally, [LIG+16] defines a mapping from LDP standard to *Constrained Application Protocol*[3] for the publication of Linked Data on the Web of Things. The current scientific works are in their infancy and targeted to specific situations and models. Therefore, we do not consider them again in this thesis.

LDP implementations can be categorized mainly in *LDP resource management systems* and *LDP frameworks* and we further describe them in Section 2.4.3 and Section 2.4.3 respectively.

LDP Resource Management System

LDP resource management systems are repositories that can host LDP resources on top which CRUD operations adhering to the LDP standard are allowed through HTTP interactions. To our knowledge, all LDP resources management systems are referenced in the LDP conformance report[4] with the exception of Cavendish[5] that also forms part of this category. In their descriptions below, we omit Gold[6], rww-play[7] and LDP.js[8] and Cavendish because they provide little to no information about their capabilities and features and to our knowledge, no other sources describe these works.

[1] https://www.w3.org/ns/ldp, last accessed 4 August 2018
[2] https://www.bugzilla.org/, last accessed 6 August 2018
[3] http://coap.technology/, last accessed on 7 August 2018
[4] https://www.w3.org/2012/ldp/hg/tests/reports/ldp.html on 19 July 2017
[5] https://github.com/cavendish-ldp/cavendish, last accessed on 10 April 2018
[6] https://github.com/linkeddata/gold, last accessed on 10 April 2018
[7] https://github.com/read-write-web/rww-play, last accessed on 10 April 2018
[8] https://github.com/spadgett/LDPjs, last accessed on 10 April 2018

The LDP resource management systems vary mainly on their supports for the different types of LDP resources and the side tools they provide to facilitate their usage. The OpenLink Virtuoso server[1] supports only LDP RDF sources and basic containers. It supports LDP interactions to these two types of resources by using WebDAV as a proxy mechanism.

Compared to Virtuoso, Apache Marmotta additionally support LDP Non-RDF sources and is open source. It has a modular architecture consisting of a number of modules that offers additional support for managing LDP resources. It comes with a triplestore (KiwiTriplestore[2]) and support querying through SPARQL. It allows traversing RDF graph using the language LDPath [SBK+12] that is also called the XPath of Linked Data. Also, it supports versioning and reasoning.

Callimachus[3] is also open source and like Apache Marmotta, it supports LDP RDF and Non-RDF sources with the exception of basic containers. Instead, it support indirect containers. Also, it is shipped with an RDFa templating language [BWLR12] that aims at generating web pages from RDF resources with embedded RDFa [AHSB12].

Fedora Commons is an open source repository system for managing and disseminating digital content while CarbonLDP is an LDP application server. Both Fedora Commons and CarbonLDP support all types of LDP resources. In addition to this, CarbonLDP also provides a higher level API on top of LDP that is used to authenticate and manipulate resources.

None of the above LDP resource management systems provide automated support for generating LDPs even if the data is already in RDF. To generate LDP resources from existing data sources and deploy them on LDP resource management systems, manual development of LDP resource generators is required. Naively developing these generators by hardcoding design decisions in them may make the design tightly coupled with the implementation. Consequently, it may be both difficult to maintain and reuse the design. Also, further manual development may be required in LDP resource generators to generate LDP resources from data that is heterogeneous or has hosting constraints.

LDP Frameworks

LDP frameworks include APIs and code libraries for that development of components to facilitate the development of LDP applications either on the client or server side. Below, we describe two such software applications both of which are referenced by the LDP conformance report.

LDP4j [EGMGC14] is an open source Java-based framework for the development of read-write LDP specification that supports all types of LDP resources. Its aim is to abstract the complexity of the LDP protocol by providing components that can be used by clients and server applications to handle and implement LDP interactions.

Compared to LDP4j, Eclipse Lyo[4] has two main differences. Firstly, it supports only LDP basic and direct containers and non-RDF Sources. Secondly, it implements *Open Services for Lifecycle Collaboration*[5] specifications whose aim is to facilitate

[1] http://virtuoso.openlinksw.com/
[2] http://marmotta.apache.org/kiwi/, last accessed 6 August 2018
[3] http://callimachusproject.org, last accessed on 10 April 2018
[4] https://wiki.eclipse.org/Lyo/LDPImpl, last accessed on 11 April 2018
[5] https://open-services.net/, last accessed on 11 April 2018

data interoperability and reduce incompatibilities between softwares and is aimed to be independent of domains, software vendors or specific product.

The aim of LDP frameworks is only to facilitate the manual development of LDPs and thus provide no automated support for generating LDPs. Naively doing so by hardcoding design decisions in them may make the design tightly coupled with the implementation of the LDP and consequently make it both difficult to maintain and reuse the design. Moreover, LDP frameworks provide no support for handing heterogeneous data or those having hosting constraints and consequently, these have to tackled programmatically.

2.5 Synthesis

In this chapter, we have shown that Semantic Web standards can be a good alternative for enhancing data interoperability. More precisely, we have describe how RDF, RDFS/OWL and the LDP standard can enhance interoperability at syntactic, semantic and data access in highly decentralized information ecosystem. For every of these standards, we have briefly described some existing technologies and tools that facilitate their implementation and/or usage. In particular, we have stressed on the LDP standard as it is the core of our thesis and we use it as the final step when Semantic Web standards and technologies are used for publishing data.When presenting LDP related work, we showed that the current scientific literature on LDP is limited and also its referenced implementations are basic. We have categorized these implementations into LDP resource management systems and LDP frameworks.

As we have seen, with both categories of LDP implementations, setting up an LDP is still complex even if the data source is in RDF. Moreover, it requires manual development that if naively undertaken can make the design tightly coupled with the implementation. Also, if the data source is heterogeneous or have hosting constraints, further manual development would be required. The difficulty posed by existing LDP implementations can demotivate the adoption of the LDP standard. Therefore, to motivate its usage in highly decentralized information ecosystems, we require that an approach or tool for generating LDPs should satisfy the requirements (in boldface):

- **Automated LDP generation**: provide support for automating the generation of LDPs ;
- **Design Reusability**: ensure that LDP design are reusable;
- **Hosting constraints**: provide support for handling data having hosting constraints;
- **Data Heterogeneity**: provide support for handling heterogeneous data sources.

Considering the above requirements, we proceed with the next chapter where we provide a model-driven engineering approach that satisfies the latter requirements.

Part II

LDP Generation

Chapter 3

Model-Driven LDP Generation

In the previous part, we have seen that Semantic Web technologies satisfy the requirements for enhancing interoperability in highly decentralized information ecosystem. Yet, instantiating LDPs to expose existing data sources may involve different problems and we have identified requirements that a solution must satisfy to solve these problems. Therefore, in this part, we present a solution that satisfy these requirements in the form of a workflow whose core is a language to describe the design of LDPs.

In this chapter, we focus on the workflow and describe the principles and decisions on which it is based on. To this end, in Section 3.1, we illustrate the problems with current LDP implementations identified in the previous chapter when developing LDPs to expose existing data sources. Also, we describe principles from model-driven engineering that may be used to tackle these problems. Then, in Section 3.2, we provide a generalized model-driven LDP generation life cycle that is obtained by applying the latter principles in the LDP development life cycle. Moreover, we identify core LDP design and deployment aspects that may be considered in the model-driven LDP generation life cycle. Finally, in Section 3.3, we describe an LDP generation workflow that is based on the model-driven LDP generation life cycle and the identified LDP design and deployment aspects that we consider.

Contents

3.1 Foundations

In this section, we describe the foundations for automatizing the generation of LDPs from existing data sources. More precisely, in Section 3.1.1, we describe the development of LDPs for exposing data sources in the form of a life cycle in which we also illustrate the problems with current LDP implementations described in Chapter 2 (in Section 2.4.3 and 2.4.3). Then in Section 3.1.2, we describe principles from model-driven engineering that may be applied in the LDP development life cycle to solve the latter problems.

3.1.1 LDP Development

Any development process consists of three main phases namely design, implementation and deployment. In this section, we describe the development with respect to these phases required for exposing existing data sources via LDPs using existing LDP implementations wherever possible. To this end, in Section 3.1.1, we describe the parts from the architecture of an LDP. Then, we describe the phases namely design, implementation and deployment of these parts in Section 3.1.1, Section 3.1.1 and Section 3.1.1. When describing the phases, we illustrate the problems identified in Chapter 2 (Section 2.5) and use italicized text to refer to them.

LDP Architecture

Figure 3.1 shows a high-level view of an LDP from the perspective of the model-view-controller architectural pattern consisting of three components: model, view and controller. The view acts as an interface between the LDP and its clients. It routes input request from clients to the controller and returns data from the controller to clients. On obtaining a request, the view delegates it to the controller that processes it. Request processing may involve applying some control on the input (e.g. validation) and/or requesting data from the data store. The result of the processing is embedded in an LDP response and sent as output.

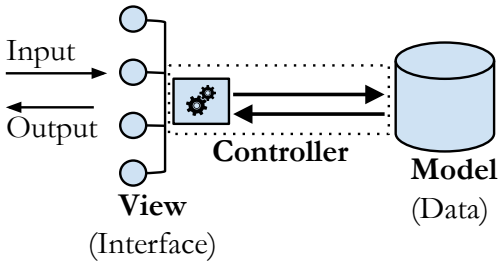

Figure 3.1: Components of an LDP

In the next three section, we describe the different phases in the development of LDPs based on the model-view-controller architecture. This description is based on our experience of developing the smart city artifacts web portal [BBZ16] that is also a linked data-based platform architectured per the model-view-controller pattern.

Design Phase

We use Figure 3.2 to describe the different phases of development. In the design phase, during *Design Decision Making*, design aspects about the data, view and controller are characterized and decisions about them are taken.

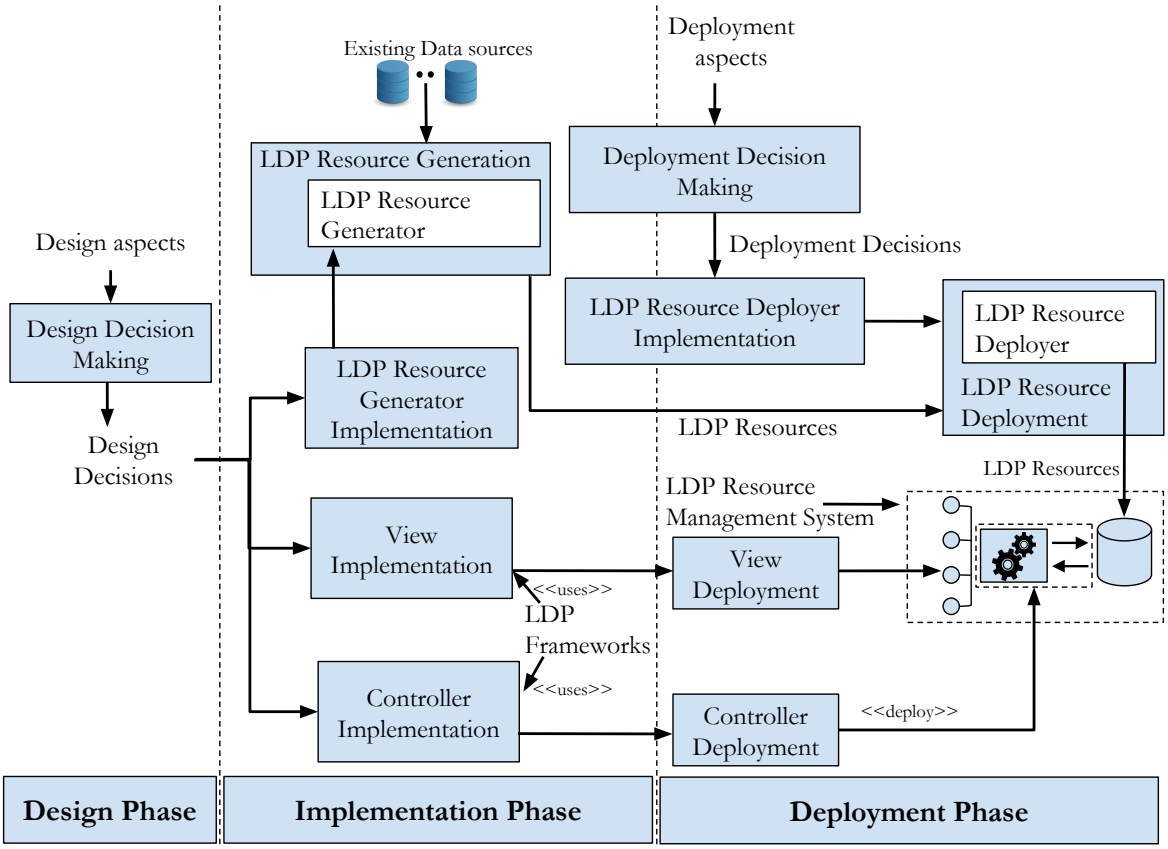

FIGURE 3.2: Overview of phases in the development of LDPs

The final output of data design is to have an LDP domain model per which data is organized on the LDP and a data structure to store the data. However, the data may be heterogeneous. Since we use RDF as a pivot data syntax, as mentioned in Chapter 2 (Section 2.1), RDF view are created for heterogeneous data by writing transformation documents that may be in languages or technologies described in Chapter 2 (Section 2.1.2). Before doing so, there may be several other sub-phases, such as design of IRIs for resources from the heterogeneous data or design of ontologies for structuring the data. While these sub-phases are fundamental to Linked Data publication, in this thesis, we reuse existing work and assume that RDF views may be created for heterogeneous and therefore consider LDP design only from RDF data.

Based on the latter assumption, data design would then involve setting up the LDP model that would normally be an instance of the LDP domain model shown in Figure 2.3. In short, this requires taking decisions about IRI and content of containers and their members. Finally, the data structure to store data need to be identified after which the different aspects from the LDP domain model would be mapped to the data structure.

Designing the view consists of two parts. Firstly, the different endpoints to expose and their structure, as they may be nested, must be defined. Secondly, the users' interactions with the endpoints need to be specified. Since LDP interaction is based on HTTP, doing so requires qualifying the design of HTTP requests and responses. Although much of this has already been specified by the interaction model of the LDP standard, there are still aspects that are open such as authentication, access rights, etc.

Finally, controller design is concerned with the design of algorithms that processes the inputs and generates the outputs. The main design aspects of these algorithms would include controls to be applied on inputs (e.g. validation, security) from the view, and the type of data required from the LDP data store, and finally the transformation to be applied on them to generate the output.

Implementation Phase

In the implementation phase, the LDP is built by encoding the design decisions in it. More precisely, with respect to the model, the LDP model is physically implemented in a data structure considering only decisions and mapping between them that were specified in the design phase. Then, an *LDP Resource Generator* is manually implemented by encoding data design decisions in it to generate LDP resources from existing data sources per the LDP model and store them in the data structure. As mentioned before, data may be heterogeneous and before they can be consumed by the *LDP Resource Generator*, further manual implementation may be required to convert them into RDF. To deploy the LDP resources, we use an *LDP Resource Deployer* whose implementation requires encoding the deployments decision taken in *Deployment Decision Making*. These deployment decision may be taken at anytime before the deployment phase.

With respect to the controller and the view, there are two ways of obtaining them. Off-the-shelf LDP controllers may be obtained by instantiating *LDP Resource Management systems* that may be thought of as consisting of a generic controller and view. In some specific situations, LDP resource management system may be too generic. In these situations, in *View Implementation* and *Controller Implementation*, custom views or controllers may be manually implemented from scratch by encoding their design decisions in their implementation. To facilitate their implementation, *LDP Frameworks* may be used.

As we have seen, developing the view, controller and LDP resource generator may require much *Manual Implementation*. Also, their development may raise the *Tight Coupling* problem as while encoding the design decisions in them, if they are tightly coupled with their implementation, it may be difficult both to maintain and reuse the design. These two problems also applies when implementing the LDP resource deployer and encoding the deployment decision in it. Moreover, as we have seen, heterogeneity of existing data sources may raise the *Data Heterogeneity* problem that may in turn again raise the *Manual Implementation* problem.

Deployment Phase

Finally, in the deployment phase, the view, controller and the data are deployed. Deploying the view and controller may be rather straightforward and done by

uploading and configuring implementations in the production environment.

To deploy the data, the LDP resources may be deployed using the *LDP Resource Deployer*. However, the data sources from which the LDP resources have been generated may have hosting constraints, such as storage limitations or license restrictions requiring. Thus, further manual implementation may have to be undertaken when exploiting these data sources. For example, in the latter case, LDP resource synchronizer may be developed to ensure that the LDP resources exposed by the LDP system is up-to-date. As we can see, in the deployment phase, the *Hosting Constraint* problem related to data sources may in turn again raise the *Manual Implementation* problem.

As we can see, the development of LDPs to expose existing data sources may raise problems especially in the implementation and deployment phase. Let us consider model-driven engineering principles that may be used to tackle these problems.

3.1.2 Model-Driven Engineering Principles

In this section, our objective is to describe model-driven engineering principles that may be used to automatize the generation of LDPs from existing data sources. To this end, in Section 3.1.2, we provide an overview of model-driven engineering methodology. Then, in Section 3.1.2 and Section 3.1.2, we describe components of model-driven engineering at the application domain and application level respectively.

Overview

Figure 3.3 shows a general overview of the core concepts in model-driven engineering and their inter-relationships. In our description, italicized words refer to concepts from this figure. In short, model-driven engineering involves using models (i.e. *Source Models* and *Object Models*) as first-class entities and transforming models to models or models to platforms using generators or by dynamically interpreting the models at run-time [FR07]. Doing so enables separation of concerns thus guaranteeing higher reusability of systems' models [SVB+06].

Two levels are identified: *Application Domain* level where *Domain Modeling Languages* and *Transformation Definition* are setup and *Application* level where models are defined using these languages and transformations. At the *Application* level, source models are created or written using domain modeling languages and automatically transformed into object models, through *Model Transformation*. They may then be deployed on platforms.

For this to be possible, at *Application Domain* level, the domain modeling languages, transformations and platforms for writing, processing models and hosting models need to be defined respectively. There exist other meta levels for defining formalisms in which domain modeling and transformation languages may be defined. However, we do not consider them as general-purpose languages (e.g. UML, OWL, Java, Python, etc.) that may be used for defining domain modeling languages and transformation already exist.

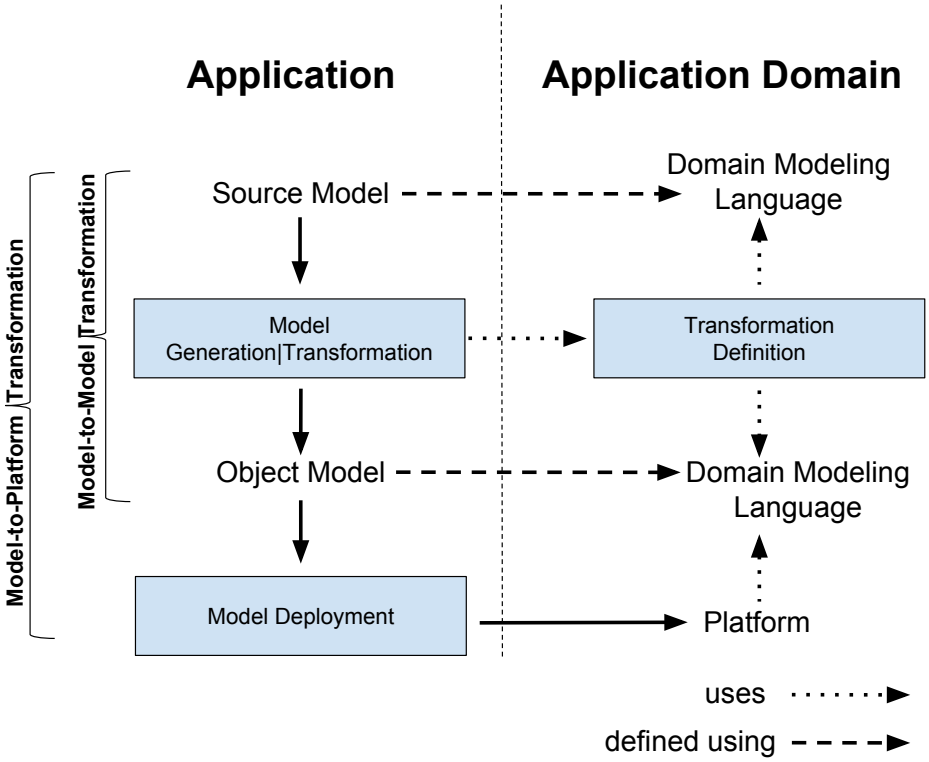

FIGURE 3.3: Application of model-driven engineering methodology adapted from [BCW12]

In this chapter, we are focus on the application level while the next chapter is focused on the application domain level. We now describe *Application Domain* and *Application* levels in more detail in Section 3.1.2 and Section 3.1.2 respectively.

Application Domain Level

In this level, the problem and solution space of the application domain are studied to identify the features participating in the *Domain Modeling Language*, *Transformation Definition* and *Platform*. We detail *Domain Modeling Language*, *Transformation Definition* and *Platform* below.

Domain Modeling Language Figure 3.4 shows a general overview of concepts in a modeling language whose aim is to formally express the relevant concepts of a domain as formal models. As such, the domain modeling language is itself based on a *meta-model* that defines valid inter-relationships between aspects of a domain that encompasses its *abstract syntax* and *static semantics*. For writing *formal model*, there may be several *concrete syntaxes* in textual or graphical representations. The *formal models* are expressed in the *concrete syntax* and are conceptually instances of the *domain modeling language*'s *meta-model*. The *formal models* gets their meanings from the domain-specific language's *semantics*.

As shown in Figure 3.5, there are two main types of domain modeling languages: configuration parameters and graph-like languages. Configuration parameters are used when domain objects have simple structures and can be described using atomic parameters while graph-like languages are used for domain objects with graph-like structures. Along the continuum between these two extremes may

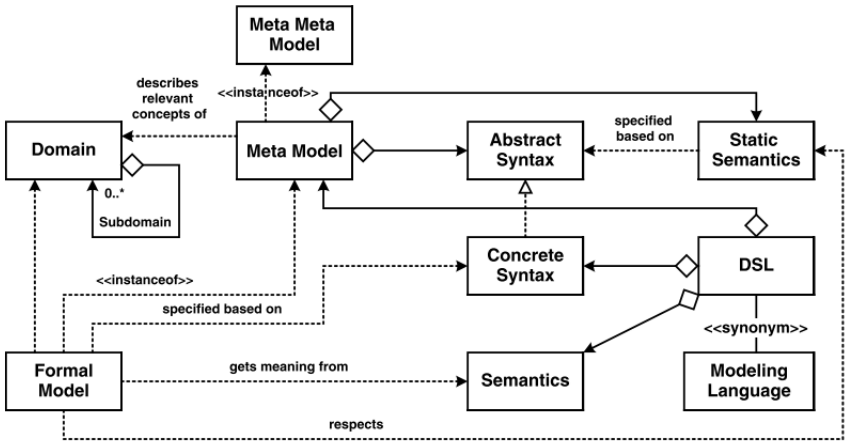

FIGURE 3.4: Overview of Domain-Specific Language in MDE [SVB+06]

lie other types of domain-specific languages such as tabular or feature-model based configuration [SVB+06].

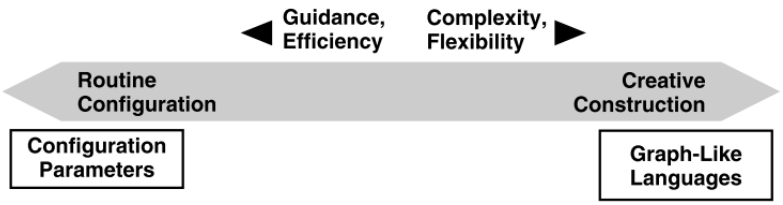

FIGURE 3.5: Main types of domain-specific language [SVB+06]

As we seen in Chapter 2 (Figure 3.2), during LDP design, design aspects may be characterized and decisions may be taken for them. In this context, a domain modeling language may be set up with constructs for characterizing these design aspects in the form of models. To transform these models into other models or executable artifacts, transformation needs to be defined. We describe Transformation Definition in the next section.

Transformation Definition As defined in [KWB03], a *Transformation Definition* "*is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language*". Also, a transformation rule "*is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language*". The source and target languages refer to the domain modeling languages in which the source and target models have been written respectively.

In the context of LDPs, transformation may be defined to generate an LDP controller in the form of executable code from the design model mentioned in the previous section. To do so, transformation rules need to defined between the constructs of the model's language and the constructs of executable code's programming language.

Platform Platform represent the ultimate solution space where models are executed or interpreted at runtime. They provide an environment in which models can be used to instantiate a running system or part of a system. Figure 3.6 shows an abstract view of a platform. As we can see, a platform supports the realization of a domain [SVB+06]. It can have several building blocks that may be built using *libraries*, *frameworks*, etc. In our context, a platform can be seen as any environment on which an LDP can be instantiated such as an LDP resource management system that was described in Chapter 2 (Section 2.4.3) or simply an LDP server.

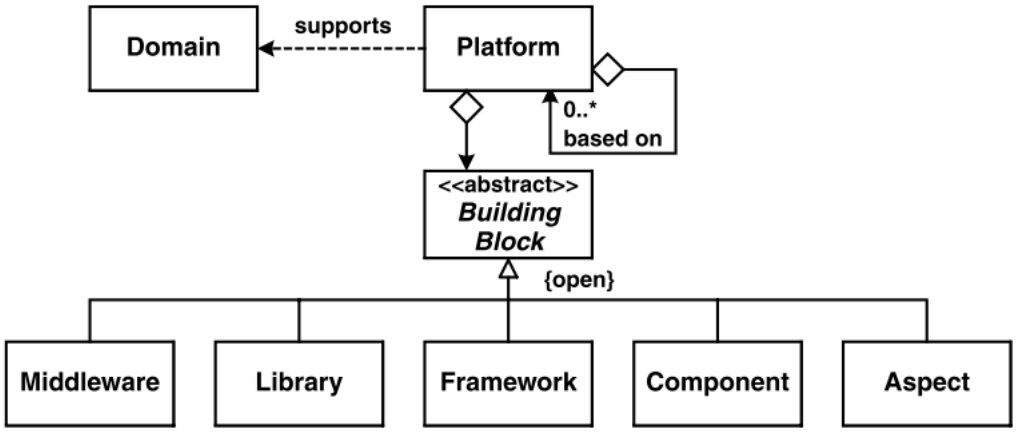

FIGURE 3.6: Abstract view of a platform [SVB+06]

Application Level

At the application level, target models for instantiating platforms partly or wholly are created from source models through model transformations. Source models may be manually written or generated from other models transformations. Thus, the focus in this level is the usage of *Model Generation/Transformation*. In model-driven engineering, a transformation "*is the automatic generation of a target model from a source model, according to a transformation definition*" [KWB03]. There are two main types of model transformations [SVB+06]: model-to-model and model-to-platform. Both model-to-model and model-to-platform transformation are based on *Transformation Definition* that as we mentioned in the previous section is a set of transformation rules between constructs of *Domain Modeling Languages*.

A *Model-to-Model Transformation* generate a target model from a source model. The source and target model are normally based on different meta-model and the latter transformation uses the mapping between the constructs of the source and target meta-model to produce the target model.

A *Model-to-Platform Transformation* is a specific type of model-to-model transformation where the target model is compatible with the target platform and can be deployed directly on it. The target model from this transformation is usually referred as generated artifacts as they can include source code or text. Also *Model Deployment* simply involve uploading the target model on the platform. This process may be part of the model-to-platform transformation itself.

For example, a model-to-model may generate LDP resources in a pivot model taking as input an LDP design model and existing data sources. However, a particular

LDP resource management system (described in Section 2.4.3) such as Apache Marmotta may use a different model for storing LDP resources. To deploy the LDP resources to Apache Marmotta, a model-to-platform transformation may be used to generate LDP resources from the pivot model to the model used by Apache Marmotta.

3.2 LDP Generation Principles

In Chapter 2 (Section 3.1.1), we have described the development required for exposing existing data sources via LDPs and highlighted the manual tasks involved. In this section, in Section 3.2.1, we describe how model-driven engineering principles, presented in Section 3.1.2, can automatize this development process. Then, in Section 3.2.2 and Section 3.2.3, we describe the design and deployments aspects of LDPs that should be considered when automatizing their generation.

3.2.1 Model-Driven LDP Generation

Figure 3.7 shows the application of model-driven engineering principles in the LDP development life cycle in Figure 3.2 at the application level (Section 3.1.2). This is why in the figure, we assume the existence of the domain modeling languages are in boldface text, and generators are in white squares, that are elements of the application domain level (Section 3.1.2) discussed in the next chapter.

In the design phase, the *LDP Design Language* is used to formalize the LDP design (described in Section 3.2.2) and their decisions in *LDP Design Models*. As mentioned before, the LDP design consist of design aspects related to view, controller and model and therefore the LDP design language provide constructs to characterize these aspects.

Also, we have a *Generation Phase* instead of an *Implementation Phase* as in Chapter 2 (Figure 3.2). In the *Generation Phase*, there three generators used. The *LDP View Generator* and *Controller Generator* exploit the LDP design model and generate the view and controller. Both processes are model-to-platform transformations as their output (i.e. view and controller) are components of the LDP.

The *LDP Resource Generator* generates LDP resources and stores them in a structure that we refer to as the *LDP Dataset* by processing the *LDP Design Model* with respect to existing data sources. The *LDP Dataset*'s meta-model is the LDP's domain model presented in Chapter 2 (Figure 2.3). Thus, this is a model-to-model transformation. However, it can also be a model-to-platform transformation if ever there is an LDP server that can directly consume the *LDP Dataset*.

Finally, during the *Deployment Phase*, the view, controller and data are deployed. While the deployment of the view and controller may be straightforward, that of the data may involve considering several deployments aspects (described in Section 3.2.3). The *LDP Deployment Language* is used to capture these decisions in *LDP Deployment Model*. This model together with the *LDP Dataset* is then used by the *LDP Dataset Deployer* to deploy the LDP resources on an LDP. This is an example of *Model Deployment* presented in Figure 3.3 and described in Section 3.1.2.

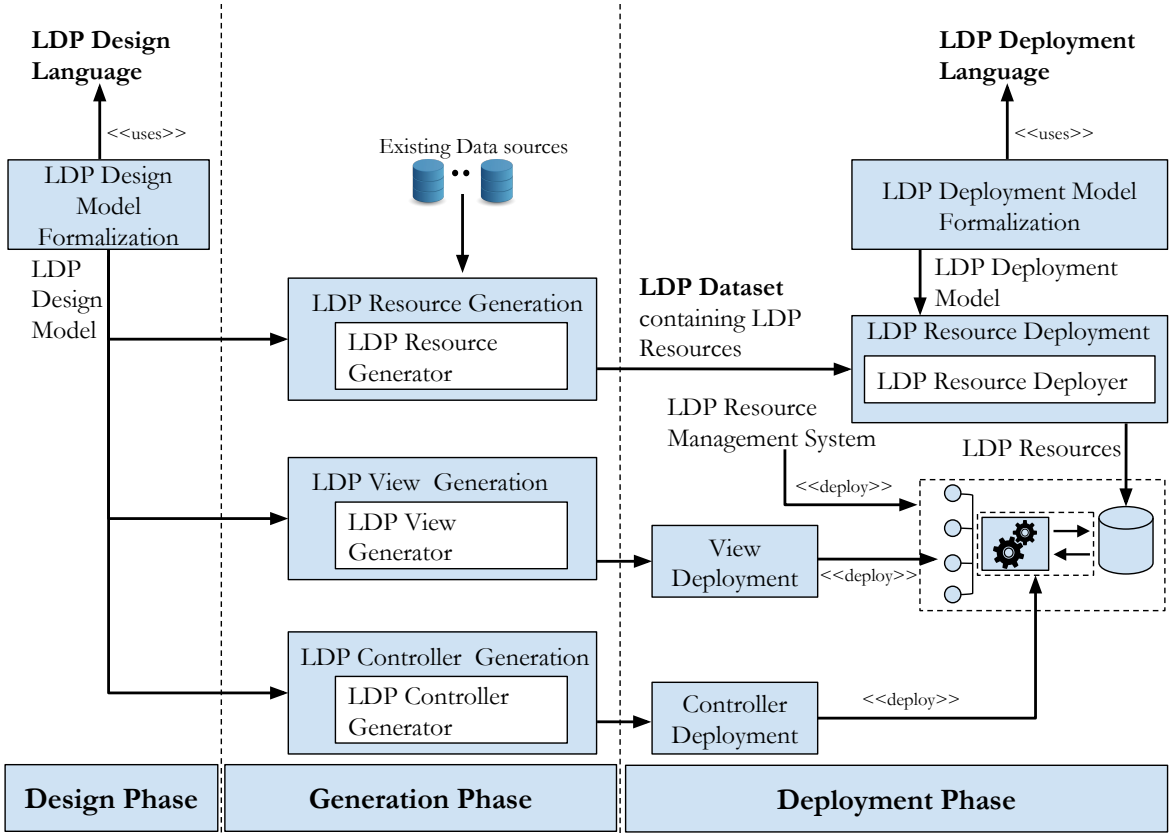

FIGURE 3.7: LDP Design and Deployment using MDE

As we can see, both the LDP design language and deployment language are central to the automatic LDP generation. All the generators from Figure 3.7 implement the semantics of either of these languages to process models written in them. Languages and generators are elements from the application domain level and as we mention before, their formalization are described in the next chapter.

Before that an LDP design or deployment language may be formalized, the aspects that they characterized needs to be identified. We identify LDP design and deployment aspects that may be characterized by an LDP design or deployment language in Section 3.2.2 and Section 3.2.3 respectively.

3.2.2 LDP Design Aspects

As we have seen, an LDP design include the design of its model, view and controller. For each of these components, several design aspects have to be considered. It may not be possible to consider all the design aspects as several of them may arise from specific use cases.

LDPs are data-driven systems and their controllers and views may entirely driven by the model (or data). Thus, it is possible to limit the LDP design only to model design and implement both the controller and view only by considering the model design. Nevertheless, the model itself may have several design aspects but we believe the fundamentals ones to be the three main attributes of an LDP resource, which is its IRI, content and type. These attributes can be naturally derived from the LDP standard.

For example, consider a GET request on an LDP Resource. To perform such a request, the resource' IRI should be known. Now, if the request is successful, the response obtained should contain at least the content of the resource that must obligatory be either in RDF or Non-RDF. Furthermore, if the content is in RDF, as per the LDP standard [SAM15c, §5.2.1.4], the result should contain an HTTP Link header indicating the type of container.

Now that we have identified the core design aspects, we describe an example in Section 3.2.2 and use it to illustrate the core design aspects *type*, *content* and *IRI* of LDP resources in Section 3.2.2, Section 3.2.2 and Section 3.2.2 respectively.

Illustrative Example

Figure 3.8 (a) shows part of an RDF graph that uses the DCAT vocabulary [ME14a] and which describes some catalogues and their datasets, distributions and themes. We use an RDF graph because as mentioned in Chapter 2 (Section 2.1), RDF satisfies the requirements for syntactic interoperability.

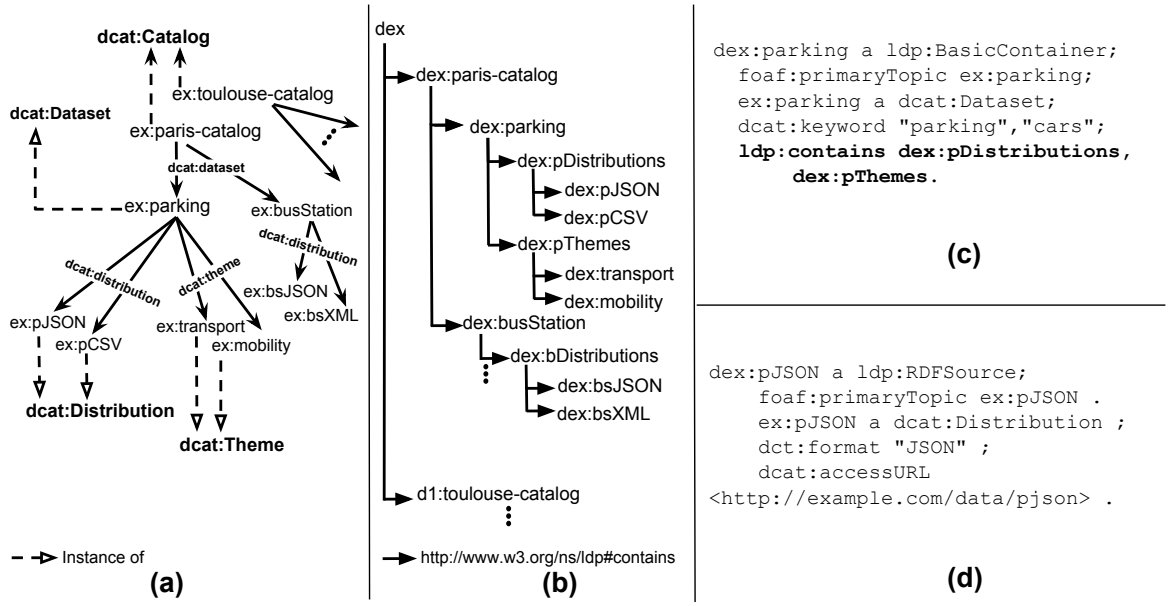

FIGURE 3.8: Example of an RDF Graph, LDP generated from it and content of some LDP resources

Figure 3.8 (b) shows the structure of an LDP generated from the RDF graph. The LDP resources describes resources from the RDF graph. Resources from the RDF graph cannot be directly used as LDP resources as the LDP standard states that "a contained LDPR[1] cannot be created (...) before its containing LDPC[2] exists" [SAM15a, §2]. Thus, for each resource from an RDF graph that needs to be exposed via an LDP, a distinct LDP resource needs to be created to describe it. The RDF resource for which an LDP resource is generated is referred to the LDP resource's *related resource*. For example, the related resource of `dex:paris-catalog` and `dex:parking` is `ex:paris-catalog` and `ex:parking` respectively.

[1]LDP Resource
[2]LDP Container

Figure 3.8 (c) and Figure 3.8 (d) shows the content of resources dex:parking and dex:pJSON respectively. Every resource from the LDP structure in Figure 3.8 (b) having an outgoing arrow is an LDP container and the resource to which the arrow points is its member. For example, the container dex:parking has members dex:pDistributions and dex:pThemes. Also, Figure 3.8 (c) shows the content of dex:parking and we can see the enumeration of these members in boldface text.

Now that we have described the example in Figure 3.8, let us use it to illustrate how the core design aspects (Type, content and IRI of LDP resource) can be characterized in the next three sections.

Type of LDP Resource

As mentioned in Chapter 2 (Section 2.4), there may be different types of LDP resources. In this work, we consider only RDF sources and basic containers. Therefore, for each resource that needs to exposed via an LDP, it must be decided whether to create an RDF source or basic container to describe it. This decision depends entirely on specific use cases. Indeed, opting for a basic container may be more flexible as it opens the possibility for other resources to be added into it. This is why some LDP resource management systems like Fedora Commons uses only containers rather than RDF sources. However, in some cases, such flexibility may not be desired.

For example, the decision has been made to create a container dex:pDistributions and RDF source dex:pJSON to describe resource ex:pDistributions and ex:pJSON from the RDF graph in Figure 3.8 (a) respectively. Such a design decision may be taken simply because the designer may intend to add more resources in dex:pDistributions in the future while ex:pJSON would not contain resources. Also, as we mentioned before, the data design may affect the view design also. Therefore such a decision may have been taken from a view perspective to provide a view design that may be more natural to the particular domain, in this case, DCAT datasets. Thus, the LDP resources type may have a considerable impact on the resulting LDP design as it may both affect the way LDP resources are organized in the data store as well as the design of the view.

Content of LDP Resource

The content of a non-container include its description in RDF. For example, dex:pJSON content in Figure 3.8 (d). On the other hand, the content a container may be seen as consisting of its description as well as its member resources. For example, consider the content of dex:parking in Figure 3.8 (c). The content of an LDP RDF resource may consist of two parts: the resource's description and the resource's members (in bold) if it is a container.

The description of a container or non-container may be obtained by considering the description of its related resource. However, RDF graphs are simply set of triples and per the RDF standard, a resource in an RDF graph does not explicitly has a subset of these triples reserved for its description. For example, consider dex:parking from Figure 3.8 (b) whose related resource is ex:parking from Figure 3.8 (a). ex:parking is simply a resource used in some triples of the RDF graph. There is no explicit set of triples that is reserved for its description. Therefore, when creating LDP RDF

sources for resources from an RDF graph, the set of triples to use as the description of the RDF resources needs to be explicitly specified.

With respect to member resources, organizing them in containers relates to a typical problem of resource organization in RESTful Web services for which there are no standard way. The organization of members in containers is an important aspect of the content of LDP resources as it directly affects the design of LDPs' view as the containers and members ultimately becomes endpoints on the views. [All10, §2.3] proposes a solution to group similar resources in containers based on application specific designs. Following this proposal, the LDP design language should provide enough freedom to allow specifying application specific designs.

For example, consider the container `dex:paris-catalog`. Its member resource includes `dex:parking` and `dex:busStation` because their related resources are linked to the related of `dex:paris-catalog` via the `dcat:dataset` as shown in Figure 3.8 (a). A different design decision could have been taken to have other resources as the member resources of `dex:paris-catalog`. For instance, resources linked via the `dcat:distribution` could have also been considered.

IRI of LDP Resource

When creating LDP resources, their IRIs have to be characterized. Even if IRIs are opaque resource identifiers, [DD11, §2] considers the design of IRIs as the most important step when in Linked Data publication.

To this end, they provide a collection of design patterns. Below, we describe the use of two of these design patterns, *patterned IRIs* and *natural keys*, that may be used when characterizing the IRIs of LDP resources.

Patterned IRIs are used to create predictable and human-readable IRIs that are both easy to remember and to create new ones. It can be applicable for sets of resources that form natural hierarchies. In our case, this pattern may be used to reflect the hierarchical relationship between LDP containers and its members. For example, let us consider creating an LDP resource having as its related resource `ex:parking` in the container `dex:paris-catalog`. When generating the IRI of the LDP resource, part of `ex:parking`'s IRI may be used together with the IRI of `dex:paris-catalog` to indicate a container-member relationship. Assuming that the expanded form of `ex` is `http://example.com/` and that of `dex` is `http://data.example.com/`, the final IRI generated for the new LDP resource may be `http://data.example.com/paris-catalog/parking`. Following this pattern may help to indicate that the resource with the latter IRI is a member of `dex:paris-catalog`.

Natural Keys are used when resources already have unique identifiers. Using this design pattern, IRIs of LDP resources can be generated by concatenating their identifier with a base. For example, let us again consider creating an LDP resource having the related resource `ex:parking`. Assume that its IRI expanded form is `http://parkings.anvers.com`. Suppose the namespace of the LDP is `http://data.example.com/`, `http://data.example.com/parkings.anvers.com`

is the new LDP resource's IRI obtained by concatenating its fully qualified domain name with the namespace.

3.2.3 LDP Deployment Aspects

In this section, we discuss the aspects to be considered when deploying LDP resources on an LDP server. In Section 3.2.3, We discuss the deployment of LDP resources on physical servers and in Section 3.2.3, we describe access rights on LDPs. In this thesis, we mostly focus on LDP design aspects when automatizing the generation of LDPs from existing data sources. Therefore, we only describe the deployment aspects below briefly.

Physical LDP Servers

When deploying LDP resources, logically they may pertain to a single LDP. However, physically, they may be stored on different LDP servers due to reasons such as storage constraints, resources partitioning or replication schemes. For an LDP resource deployer to be able to deploy the LDP resources on the appropriate physical servers, a description should be supplied to it containing the location of servers on which LDP resources are to be deployed.

Such a description may be at different level of granularity. For example, it may be as simple as an associative array where key represent the IRI of an LDP resource and the value represent the IRI of LDP server where the LDP resource is to be deployed. It may also be more complex, such as for each physical LDP server, a set of constraints may be defined and LDP resources may be stored on it only if it satisfies the latter constraints. The latter description may also be used by an LDP controller to determine the physical location of LDP resources when applying a read or write operation on them.

If we consider the structure in Figure 3.8 (b), it may be seen as a logical view of an LDP. The resources from it may be partitioned on different servers. Suppose an LDP controller receives a `GET` request on the resource `dex:paris-catalog` from the structure, it may use a deployment description of resources to know the location of the resource to retrieve it and generate the response.

Access Rights

As mentioned in Chapter 2 (Section 2.4.1), the LDP standard defines several read-write operations that may be performed in LDP resources. In different use cases, such as where there is sensitive data, there is a need to restrict these operations only to authorized users. Access rights may be assigned to users or group of users to describe whether or not they are allowed to perform some operations. Such a description may also be used by the controller to apply access control on LDP request from users such as verifying if the operations on the requested LDP resources are allowed or not.

Considering the LDP structure in Figure 3.8 (b), different type of access rights may be defined for the resources. For example, restrictions may be placed on containers (e.g. `dex:paris-catalog`,`dex:parking`) to allow some users to add or

delete members. Similarly, restrictions may be placed on all resources, whether containers or not, to allow certain users to update their RDF description.

3.3 LDP Generation Workflow

In the previous section, we have presented the model-driven LDP generation and taken the decision to restricted LDP design to the three core design aspects (type, IRI and content) and LDP deployment to a single LDP server. In this section, we present our final proposal that satisfies the requirements specified in Chapter 2 (Section 2.5) for facilitating the generation of LDPs from existing data sources. We refer to this workflow, shown in Figure 3.9, as the *LDP Generation Workflow* and it is based on the model-driven LDP generation together the decision taken regarding the LDP design and deployment.

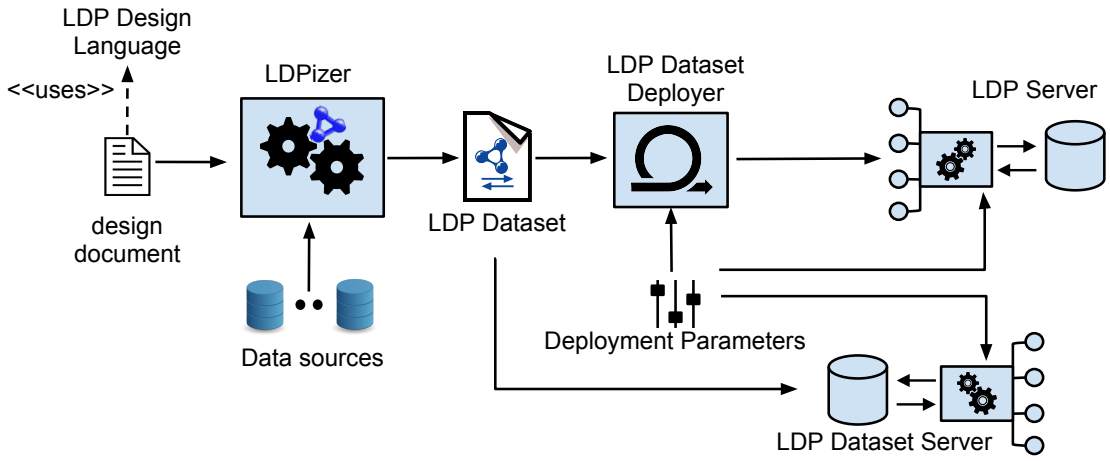

FIGURE 3.9: The LDP Generation Workflow

The LDP generation workflow consists of two main processes: *LDPization* and *Deployment*. During the LDPization process, an LDP design language is used to describe the design of the LDP in what we refer to as the *design document*. Then, the design document is used with respect to the existing data sources to generate LDP resources in a structure that we refer to the *LDP dataset*.

In the deployment process, the LDP dataset is deployed by instantiating an LDP from it. The LDPization process precedes the deployment process but may not necessarily be consecutive. The output from the LDPization process can be kept as input to the deployment process at a different time and on a different machine.

We further describe the LDPization and Deployment process in Section 3.3.1 and Section 3.3.2. In our description below, we make reference to the four requirements identified in Chapter 2 (Section 2.5) using their names[1] in italic and explain how they are satisfied.

[1]Automatic LDP Generation, Design Reusability, Heterogeneity and Hosting Constraint

3.3.1 LDPization Process

The LDPization process is performed by an *LDPizer*, that is basically a generic LDP resource generator as described in Section 3.2.1. The LDPizer takes as input some existing data sources and a design document and output an LDP dataset containing the LDP resources. We detail the LDPizer and its inputs and output below. As we will explain in the next chapter, it can deal with static and dynamic data sources.

Existing Data Sources

The existing data sources are from where the content for the LDP resources will be extracted. It can be in any data syntax but as mentioned before, we use RDF as a pivot data syntax. Therefore, all non-RDF data sources are converted to RDF using their transformation documents using techniques described in Chapter 2 (Section 2.1.2). As we will describe in the next chapter, LDP-DL provide a construct to specify transformation documents for non-RDF sources.

Design Document

As mentioned before, the design document contains the definition of an LDP design expressed in the LDP design language. As mentioned before, for now, we consider only the core design aspects and therefore the LDP design language provides constructs for characterizing only these aspects. With respect to the different types of languages described in Section 3.1.2, it has a graph-like structure as it abstractly describes the LDP resources and their inter-relationship that forms a graph as we can see in Figure 3.8 (b).

As we mentioned in Section 3.2.1, it is possible to automatically generate the design from existing data sources. However, doing so requires identifying design patterns for the core design aspects that is beyond the scope of our work. However, as we will describe later, we provide a way to partially automatize the design phase through generic designs/patterns.

The LDP design language satisfies the *Design Reusability* requirement by enabling the formalization of LDP designs in design documents. The design documents are separate and independent from any implementations may be reused by different actors. As we will describe in the next chapter, it also satisfies the *Data Heterogeneity* and *Hosting Constraint* requirements by allowing the definition of LDP design on heterogeneous data sources.

LDPizer

The LDPizer participates in the *Automatic LDP Generation* requirement by automating the generation of LDP resources in an LDP dataset from a design document with respect to to some existing data sources. As mentioned before, this generation is based on a model-to-model transformation (described in Section 3.1.2).

Moreover, as we will explain in the next chapter, the LDPizer also participate in satisfying the *Hosting Constraint* requirement by generating a variant of LDP dataset that may be interpreted at query time to exploit constrained data sources without having to host content from them.

LDP Dataset

As mentioned before, an LDP dataset is an implementation-agnostic data structure to store LDP resources. An instance of the LDP dataset is the semantics of a design document in the LDP design language. Different LDP implementations may have their own physical data structures of store LDP resources all of which can be seen as instances of an LDP dataset. In an LDP dataset, each LDP resource is assigned a URL and has an associated RDF graph, and a set of members if it is a container.

As we will explain in the next chapter, there are different variants of LDP dataset, one of which we refer to as *dynamic LDP dataset*. This variant of LDP dataset stores LDP resources and instructions to generate their content at query time and thus participates in satisfying the *Hosting Constraint* requirement.

3.3.2 Deployment Process

The deployment process involves instantiating an LDP from an LDP dataset. It can be done in two different ways based on the nature of an LDP server using some deployment parameters. Firstly, an *LDP dataset deployer* can be used to deploy the LDP dataset on an existing LDP server. Secondly, the LDP dataset can be directly consumed by an LDP dataset server. The deployment parameters required, LDP dataset deployer and LDP dataset server is described in Section 3.3.2, Section 3.3.2 and Section 3.3.2.

Deployment Parameters

In this work, we focus mostly on LDP design and therefore we perform simple deployment. More precisely, we deploy LDPs only on a single physical server that may require basic authentication. For characterizing these deployment aspects, we use simple deployment parameters that, in the context of model-driven engineering, would be considered as a simple configuration-based language which is a type of domain modeling language as described in Section 3.1.2. The deployment parameter that we consider are IRI for the location of the physical server and username and password for basic authentication.

LDP Dataset Deployer

The LDP dataset deployer contribute to the *Automatic LDP Generation* requirement by automating the deployment of LDP datasets in two different ways. Firstly, it can use a model-to-platform transformation described in Section 3.1.2 to generate from the LDP dataset an implementation-specific artifact per the LDP server's model containing the LDP resources and then upload it into the data store of the server.

Secondly, as mentioned in Chapter 2 (Section 2.4.1), LDP resources may be created on servers via LDP `POST` requests. Thus, the LDP dataset deployer may generate LDP `POST` requests for each LDP resources from the LDP dataset to deploy them on the corresponding LDP server. By doing so, the LDP server itself handles the materialization of LDP resources in its implementation-specific model.

These two techniques have some pros and cons with respect to each other. The first one may be more optimal when the LDP dataset is of considerable size as

creating POST requests for LDP resources and sending all of them at once may create a bottleneck both at the sender's site when creating them, on the network when sending them and at the destination LDP server when processing them.

The second one may be used when model used the LDP server to store LDP resources in not known such as when the LDP server is closed source. This is because to convert the LDP dataset to an implementation-specific artifact containing the LDP resources, the mapping between the LDP dataset and the model used by the LDP server must be know.

LDP Dataset Server

The LDP dataset server is a compatible LDP server that can directly consume an LDP dataset to expose resources from it. In Section 3.2.2, we explained that the controller and view of an LDP can be instantiated using only the data design aspects as LDPs are data-driven systems. Actually, the LDP Dataset Server is based on this concept. More precisely, it is a platform (described in Section 3.1.2) and consists of a generic controller and view that are developed based on the core design aspects and are instantiated the LDP dataset (the model). Thus, it participates in satisfying the *Automatic LDP Generation* requirement.

The LDP dataset server also participate in the *Hosting Constraint* requirement as it can interpret the dynamic LDP dataset mentioned in Section 3.3.1 and expose LDP resources without having to host their content.

3.3.3 Workflow Instantiation Example

Let us now instantiate the LDP generation workflow in using the example in Section 3.2.2. In the example, the LDP design language may be used to define a design model to generate the LDP in Figure 3.8 (b) from the RDF graph in Figure 3.8 (a). The model may then be expressed in the concrete syntax of the language in a design document. In this case, the data source is already in RDF. But as we mentioned, we use RDF as a pivot data syntax. Heterogeneous data sources may be considered by considering their RDF transformation that may be specified in the design document using a construct provided by the LDP design language. The LDPizer may use the design document and existing data sources to generate an LDP dataset containing the LDP resources in Figure 3.8 (b). Such an LDP dataset may then be deployed on an LDP server using either of the two ways described in Section 3.3.2. Also, an LDP can be directly instantiated by using the LDP dataset server to expose LDP resources from the LDP dataset.

3.4 Summary

At the beginning of this chapter, we have described an LDP development life cycle and highlighted problems that may be raised in it and describe model-driven engineering principles that may be used to tackle these problems. By integrating the principles in the LDP development life cycle, we have provided a generalized model-driven LDP generation life cycle whose core are languages to describe design and deployment aspects. Then, we have explained our decision to restrict the LDP

design and deployment to only some aspects. Based on the model-driven LDP generation life cycle and decisions taken regarding LDP design and deployment, we have provided an LDP generation workflow that satisfies the requirements for facilitating the generation of LDPs identified at the end of the previous chapter.

Chapter 4

LDP Design Language

In the previous chapter, we have shown that the development of LDPs can be automated using a model-driven engineering approach. More precisely, we have provided the LDP generation workflow (in Section 3.3) to automatize the generation of LDPs from existing data sources. As we have seen, a core element of the LDP generation workflow is the design document written in LDP-DL that describe the design of an LDP and using an LDP dataset containing the LDP resources are generated. To be able to write design documents and implement interpreters or transformers that can process them, the syntax and semantics of LDP-DL need to specified clearly.

Therefore, in this chapter, our objective is to formally describe the syntax and semantics of LDP-DL. To this end, in Section 4.1.1, we start with an overview of our language. Then, we formally describe its syntax and semantics in Section 4.2. In Section 4.3, we describe LDP datasets, variabilities that can affect its validity and provide ways to abstract these variabilities. Finally, in Section 4.4, we provide the operation semantics of LDP-DL in the form of algorithms and demonstrate their correcteness.

Contents

4.1 Overview of LDP-DL

In this section, we provide an overview of LDP-DL together with its syntax and semantics in Section 4.1.1 and then in Section 4.1.2, we illustrate its use in the generating an LDP using example.

4.1.1 LDP-DL Model

The core constructs of LDP-DL are `ContainerMap`, `NonContainerMap`, `ResourceMap` and `DataSource` and they are shown in its abstract model in Figure 4.1. They enable defining the core LDP design aspects identified in Chapter 3 (Section 3.2.2) with the exception of IRIs. We exclude IRIs and consider it only in our concrete model, described in the next chapter, because according to the LDP standard, the IRI of an LDP resource is entirely determined by the LDP server [SAM15c, §5.2.3.10].

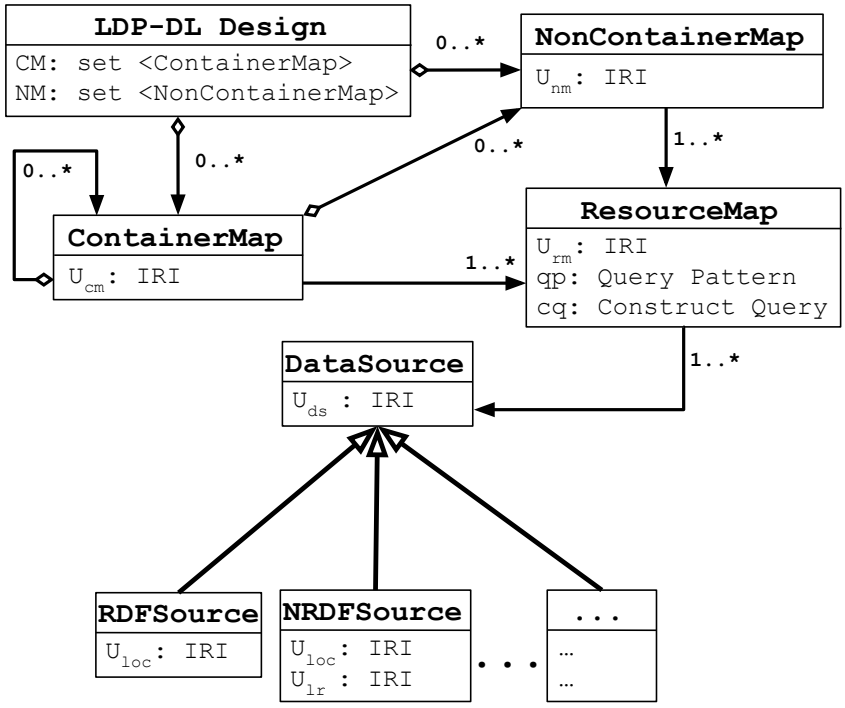

FIGURE 4.1: Abstract model of LDP-DL in UML notation.

As we can see, the constructs have an IRI that is their names. LDP-DL designs are instances of the model shown in Figure 4.1. In short, they provide an abstract description of the containers and non-containers on an LDP. `ContainerMaps` and `NonContainerMaps` are used to describe these containers and non-containers respectively. `ContainerMaps` and `NonContainerMaps` use `ResourceMaps` to specify the graph of these (non-)containers and the resources they describe. As we mentioned before, we refer to *related resource* as the RDF resource that an LDP resource describes and this is an important term that we use in our syntax and semantics. `ResourceMaps` use query patterns and CONSTRUCT queries with respect to some `DataSources` to specify related resources of LDP resources and their

graph respectively. `ContainerMaps` can be nested with other `ContainerMaps` and `NonContainerMaps` to allow for nested hierarchies of LDP resources as containers can have members that may be themselves containers and non-containers. Consequently, containers and non-containers can have ancestors and their related resources and graphs can depend on these ancestors.

Let us now illustrate the above concepts using an example in the next section.

4.1.2 Illustrative Example

Figure C.1 shows part of a design in LDP-DL. Though not shown in the figure, the `DataSource` of the `ResourceMaps` is the RDF graph in Figure 4.3 (a). Figure 4.3 is the same as Figure 3.8 in Chapter 3. We replicate the figure here for the sake of readability. Also, in Figure 4.2, an arrow with the label cm, nm or rm indicates that the language construct has a `ContainerMap`, `NonContainerMap` or `ResourceMap`. Also, arrows originating from `ResourceMaps` with the label qp and cq indicate their query patterns and CONSTRUCT queries respectively. Here, we do not provide these queries but only their description in natural language as they contain a number of concepts that we explain when providing the abstract syntax. The same LDP-DL design with the queries is provided in Appendix C.1.

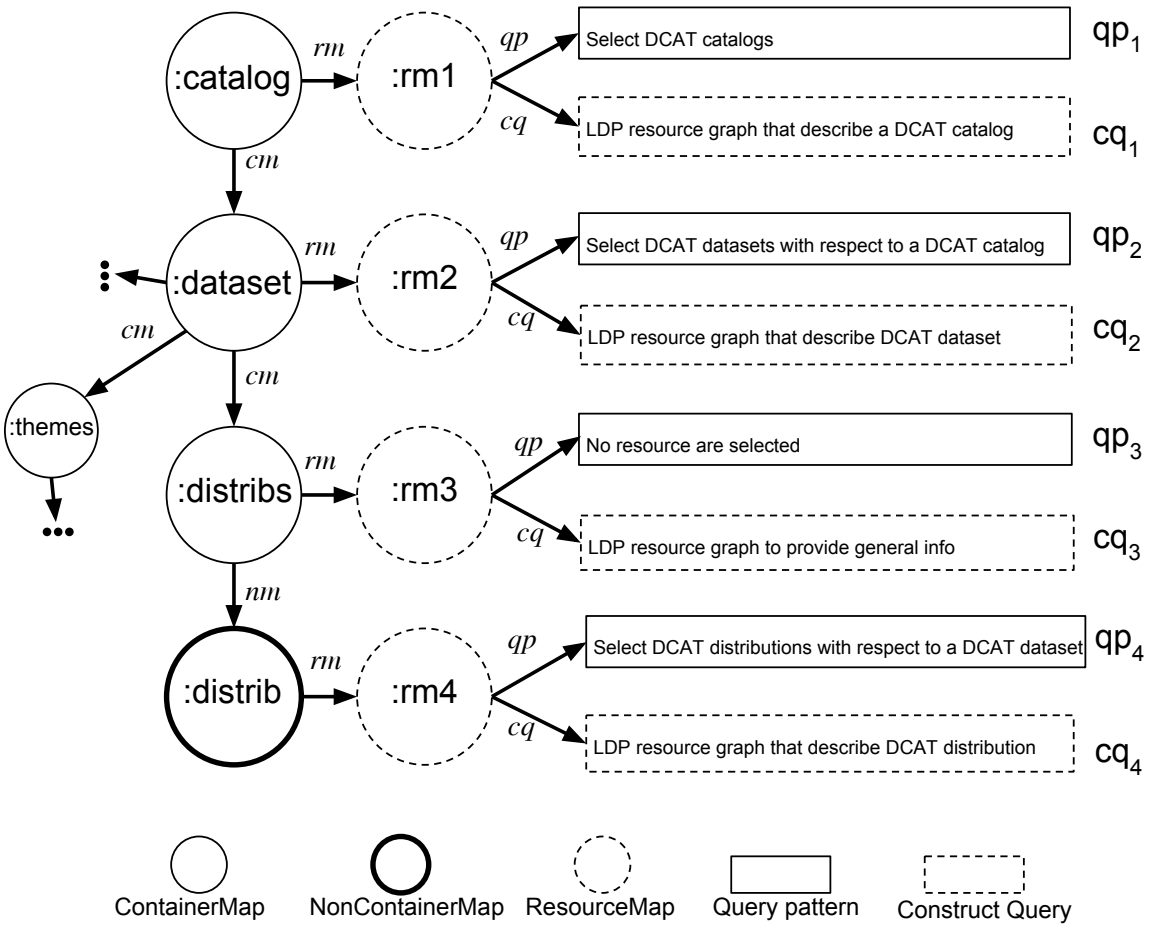

FIGURE 4.2: Part of an LDP-DL design

The RDF graph in Figure 4.3 (a) uses the DCAT vocabulary that describes data catalogs having datasets that in turn may have distributions. The design document in Figure 4.2 is used for generating an LDP[1] having a similar structure. In the design document, `:catalog` is used for creating containers for describing DCAT catalogs and in which `:dataset` in turn creates containers as members that describes the DCAT dataset of these catalogs. DCAT datasets can have themes and distributions. Rather than directly creating LDP resources for themes and distributions directly in their corresponding dataset containers, we use `:distribs` and `:themes` to create distinct containers for grouping these LDP resources. In this way, we can directly refer to the LDP resources that describes distributions or themes of a particular dataset. In this example, we consider only `:distribs` and from the containers that it generates, `:distrib` create non-containers as their members for describing the corresponding distributions.

Let us now describe the evaluation of the design document in Figure 4.2 with respect to the RDF graph in Figure 4.3 (a) as the `DataSource` to generate the LDP whose structure is shown in Figure 4.3 (b). In the description below, resources with prefixes ex, dex or `:` are from Figure 4.3 (a), Figure 4.3 (b) or Figure 4.2 respectively.

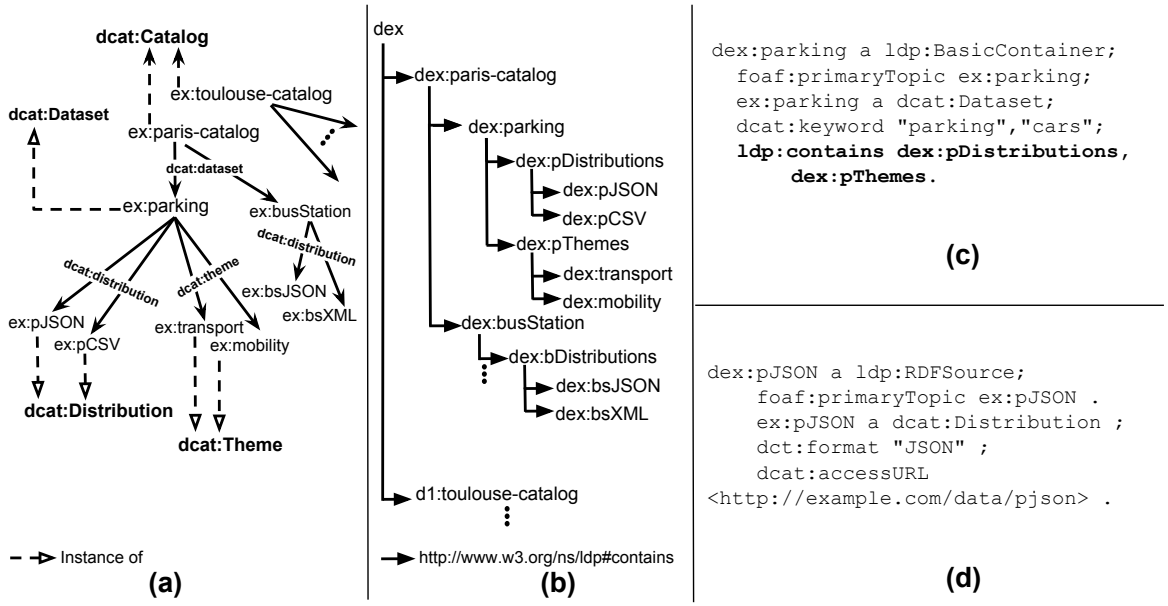

FIGURE 4.3: Example of an RDF Graph, LDP generated from it and content of some LDP resources

:catalog

The `ContainerMap` `:catalog` uses the `ResourceMap` `:rm1` to generate the top-level containers that describe the DCAT catalogs from the RDF graph in Figure 4.3 (a). The evaluation of the query pattern of `:rm1` returns related resources `ex:paris-catalog` and `ex:toulouse-catalog` that are the DCAT catalogs.

For each of these related resources, the LDP resources are generated `dex:paris-catalog` and `dex:toulouse-catalog` respectively. For both LDP resources, their RDF graphs

[1] http://opensensingcity.emse.fr/ldpdfend/catalogs/ldp,

are obtained by evaluating the CONSTRUCT query of :rm1 on the RDF graph. This evaluation can be parameterized by the related resource and ancestors' related resources. For example, the evaluation of cq1 to generate the graph of dex:paris-catalog is parameterized by its related resource ex:paris-catalog.

After the LDP resources and their graph have been generated using :rm1, they are typed as containers by :catalog. Finally, new containers generated from :catalog must define their members as well, and is thus satisfied only if their members correspond to the resources generated by their underlying ContainerMaps and NonContainerMaps which in this case, is :dataset only.

:dataset

:dataset is used to generate containers that are eventually added as members to containers generated from :catalog. Its evaluation takes place in the context of every container generated from :catalog. Let us consider its evaluation in the context of dex:paris-catalog.

As mentioned before, dex:paris-catalog describes the DCAT catalog ex:paris-catalog. At this point, the aim of using :dataset is to generate containers that describe the DCAT datasets of of ex:paris-catalog, that is the related resource of dex:paris-catalog. Consequently, the extraction of related resources using qp2 is parameterized by ex:paris-catalog, to obtain the DCAT datasets ex:parking and ex:busStation. For both DCAT datasets, LDP resources and their graphs are generated using cq2. Here, we have illustrated the use of the related resource ex:paris-catalog of the ancestor dex:paris-catalog.

Finally the the evaluation of :dataset generates two containers dex:parking and dex:busStation that are added to the members of dex:paris-catalog. The evaluation of :dataset continues with its underlying ContainerMaps and NonContainerMaps. In this case, we consider only :distribs and continue our explanation using its evaluation.

:distribs

:distribs is used to generate containers as members for containers generated by :dataset. Let us consider its evaluation in the context of dex:parking whose related resource is ex:parking.

As we have mentioned before, :distribs does not explicitly describe any resource from the data source and therefore, containers generated from it do not have any related resource. These containers are used only for grouping LDP resources. This is why qp3 only returns one empty solution mappings and consequently a single container dex:pDistributions is generated having no related resource. The graph of the container is generated by evaluating cq3. Since there is no related resource, the graph of the container may contain some general descriptions in the CONSTRUCT query. The evaluation of :distribs continues with the NonContainerMap :distrib.

:distrib

`:distrib` is used to generate non-containers as members of containers generated from `:distribs` for describing their corresponding DCAT distributions. Let us consider its evaluation in the context of `dex:pDistributions` that, as mentioned before, does not have a related resource.

In this context, `:distrib` is used for generating non-containers that describe the DCAT distributions of `ex:parking`. Notice that at this level, `dex:parking` is the grand-parent of all non-containers that will be generated by `:distrib`. Therefore, to obtain these DCAT distributions, the evaluation of `qp4` is parameterized by the related resource of grand-parent, that is `ex:parking` and the results of this evaluated are `ex:pJSON` and `ex:pCSV`. For both DCAT distributions, non-containers `dex:pJSON` and `dex:pCSV` are created and their graph generated using `cp4`. Finally, these non-containers are added as members to `dex:pDistributions`.

Now that we have an overview of the syntax and semantics of LDP-DL, let us formally describe it in the next section.

4.2 Formal Description

In this section, we formally describe the syntax and semantics of LDP-DL. We start by giving the formal preliminaries in Section 4.2.1. Then, in Section 4.2.2 and Section 4.2.3, we describe the syntax and semantics respectively.

4.2.1 Preliminaries

In this section, we give the formal preliminaries related to RDF and SPARQL that we use throughout this chapter.

RDF Let $\mathbf{IRI}$, $\mathbf{B}$, $\mathbf{L}$ and $\mathbf{V}$ be the disjoint sets of *IRIs*, *blank nodes*, literals and variables respectively. The set of *RDF terms* is $\mathbf{T} = \mathbf{IRI} \cup \mathbf{B} \cup \mathbf{L}$. A *RDF triple* is an element of $\mathbf{IRI} \cup \mathbf{B} \times \mathbf{IRI} \times \mathbf{T}$ and the set of all RDF triples is $\mathcal{T}$. An *RDF graph* $g \in 2^{\mathcal{T}}$ is a finite set of RDF triples, and the set of all RDF graphs is $\mathcal{G}$. If $\mathbf{G}$ is a set of RDF graphs, we denote the merge of all the RDF graphs in $\mathbf{G}$ as $\mathrm{Merge}(\mathbf{G})$.

SPARQL We provide an overview of part of the syntax and semantics. Common to the four query forms discussed in Chapter 2 (Section 2.3.1) is a query pattern qp (WHERE clause) that is defined recursively as follows:

1. a triple pattern $tp \in \mathbf{T} \cup \mathbf{V} \times \mathbf{IRI} \cup \mathbf{V} \times \mathbf{T} \cup \mathbf{V}$ is a query pattern;
2. a graph pattern (or graph template) $gp \in 2^{tp}$ is a query pattern;
3. if p_1 and p_2 are query patterns, then $(p_1\ \mathbf{AND}\ p_2)$, $(p_1\ \mathbf{UNION}\ p_2)$ and $(p_1\ \mathbf{OPT}\ p_2)$ are query pattern;
4. a filter expression consisting of query patterns and SPARQL built-in conditions is a query pattern (see [HS13a, §17.2] for more details about filter expressions).

The evaluation of a query pattern qp over an RDF graph G is denoted by $[\![qp]\!]_G$. Such an evaluation returns a list of solution mappings Ω. A solution mapping μ is a partial function $\mu : \mathbf{V} \rightarrow \mathbf{T}$ and its domain is denoted by $\mathrm{dom}(\mu)$.

A SPARQL CONSTRUCT query is a pair $(\mathsf{gt}, \mathsf{cq})$ where gt is a graph template and cq a query pattern. Given a graph template gt and a set of solution mappings Ω, we write $\mathsf{gt}(\Omega)$ to denote the RDF graph formed by taking each solution mapping in Ω, substituting for the variables in the graph template, and combining the triples into a single RDF graph by set union.

4.2.2 Abstract Syntax

In this section, we describe the abstract syntax of LDP-DL in a way that is used by its semantics described in the next section. As mentioned before, the LDP-DL design in Figure 4.2 with details of the query pattern and CONSTRUCT query is provided in Appendix C.1. When formally describing the LDP-DL constructs below, we illustrate them using different parts from the latter LDP-DL design. Also, in Appendix C.1, we extend the description given in Section 4.1.2 with concepts from the abstract syntax that we provide below.

Let us now describe the abstract syntax using a bottom-up approach starting with the definition of `DataSource`, `ResourceMap`, `NonContainerMap`, `ContainerMap` to finally end with a design document.

DataSource

A `DataSource` describes an RDF graph. There are several ways of doing so that our concrete syntax (Appendix B) covers. In the abstract syntax, we consider only RDF sources and Non-RDF Sources.

Definition 1 (`DataSource`). *$\langle u_{\mathsf{ds}}, u_{\mathsf{loc}} \rangle$ describes an RDF data source u_{ds} whose RDF graph is located at u_{loc}. $\langle u_{\mathsf{ds}}, u_{\mathsf{loc}}, u_{\mathsf{lr}} \rangle$ describes a non-RDF data source u_{ds} located at u_{loc} from which an RDF graph can be generated using an RDF transformation document that we refer as the* lifting rule *located at u_{lr}. u_{ds}, u_{loc} and u_{lr} are IRIs.*

As we mentioned before, the design document in Figure C.1 does not have `DataSources`. But suppose that its `ResourceMaps` have the Paris DCAT catalog[1] as their data source. Such a data source can be described as $ds_1 = \langle \texttt{ex:ds1}, \texttt{paris:ttl} \rangle$[2] with `ex:ds1` being the name of ds_1 and `paris:ttl` the location of the RDF graph.

Suppose that the data source at `paris:ttl` was not in RDF, a lifting rule provided by the data publisher at `paris:liftingRule` could have been used to generate RDF from it. Such a data source can be described as $ds_2 = \langle \texttt{ex:ds1}, \texttt{paris:ttl}, \texttt{paris:liftingRule} \rangle$[2]

ResourceMap

A `ResourceMap` has a SPARQL query pattern qp and a CONSTRUCT query cq to extract a set of related resource and generate the RDF graph of LDP resources

[1] https://opendata.paris.fr/api/v2/catalog/exports/ttl, last accessed 5 September 2018

[2] Prefix `paris` expands to `https://opendata.paris.fr/api/v2/catalog/exports`

respectively. Also, it has a set of DataSources and its both queries are evaluated on the union of the RDF graphs obtained from these DataSources.

Definition 2 (ResourceMap). *A* ResourceMap *is a tuple* $\langle u_{\text{rm}}, \text{qp}, \text{cq}, \mathbf{DS}\rangle$ *where* u_{rm} *is an IRI,* qp *is a SPARQL query pattern,* cq *is a SPARQL CONSTRUCT query, and* $\mathbf{DS}$ *is a set of* DataSources*.*

Consider the ResourceMap :rm1 in Figure 4.4. Suppose that query pattern and CONSTRUCT query of :rm4 are qp_4 and cq_4 respectively and that its DataSource is $ds_1 = \langle \texttt{ex:ds1}, \texttt{paris:ttl}\rangle$. In the abstract syntax, $rm_4 = \langle \texttt{:rm1}, \text{qp}_1, \text{cq}_1, \{ds_1\}\rangle$ describe the ResourceMap :rm4.

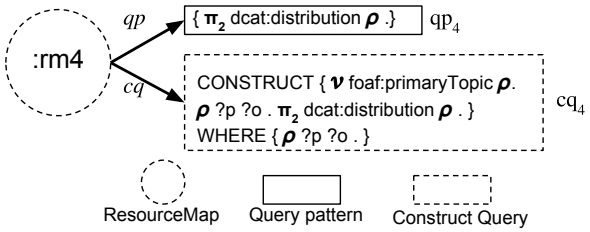

FIGURE 4.4: Example of a ResourceMap

As we can see in Figure 4.4, in the query pattern and CONSTRUCT query of :rm4, there are some characters in boldface text. These are special variables in our language. In fact, we assume the existence of an infinite set of variables $V_r = \{\rho, \nu, \pi_1, \ldots, \pi_i, \ldots\} \subseteq \mathbf{V}$ called the *reserved variables*, such that $\mathbf{V} \setminus V_r$ is infinite. ρ is used to refer to related resources (related resource variable). ν is used to refer to an LDP resource (new resource variable). It is normally used in the graph template of a CONSTRUCT query to refer to the LDP resource in the RDF graph to be generated. $\pi_1, \ldots, \pi_i, \ldots$ is the infinite set of ancestor variables to refer to related resources of ancestors. ResourceMaps may use the reserved variables but these have a special semantics as explained in the next section. However, due to undesirable consequences, we forbid the use of variable ν in the WHERE clause of the CONSTRUCT query cq. Let us describe more about the usage of these variables in the following paragraphs.

Ancestors Variables A ResourceMap is evaluated in the context of an LDP resource that may itself be within a container, its parent, or it may be a top-level resource having no ancestors. The ancestors of an LDP resource's related resource are the related resources of the LDP resource's ancestors and thus, the ancestor variables (π_1...π_n) are bounded to these related resources. For example, with respect to an LDP resource, π_1 is bounded to its parent's related resource and π_2 is bounded to its grandparent related resource. In case, an ancestor container exist without a related resource, then the corresponding ancestor variable is unbounded. For a LDP resource that is a top-level resource, all its ancestor variables are unbounded.

qp & cq of ResourceMaps In qp, ρ is unbounded and the *ancestor* variables (π_1...π_n) may be used to further constraint the extraction of related resources. The evaluation results of qp are projected on ρ and is a set of related resources. For

each of them, an LDP resource is created, and cq is evaluated with ρ bounded to the related resource and ancestor variables bounded to the ancestors of the LDP resource. cq has an additional variable nu that is bounded to the LDP resource itself.

NonContainerMap

A `NonContainerMap` has a set of `ResourceMaps` that is used for generating its non-containers.

Definition 3 (`NonContainerMap`). *A* `NonContainerMap` *is a pair* $\langle u_{\text{nm}}, \mathbf{RM} \rangle$ *where* u_{nm} *is an IRI and* $\mathbf{RM}$ *is a set of* `ResourceMaps`*.*

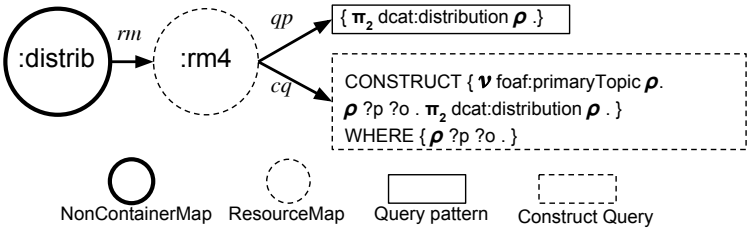

FIGURE 4.5: Example of a `NonContainerMap`

Consider the `NonContainerMap` `:distrib` in Figure 4.5. Its `ResourceMap` is the same as in the previous example. In the abstract syntax, $distrib = \langle$`:distrib`$, \{rm_4\}\rangle$ describe the `NonContainerMap` `:distrib`. Normally, `:rm4` will generate a set of LDP resources and their RDF graphs that will then be types as non-containers by `:distrib`.

ContainerMap

A `ContainerMap` has a set of `ResourceMaps` for generating its containers. To further generate containers and non-containers in them, its set of child `ContainerMaps` and `NonContainerMaps` are used.

Definition 4 (`ContainerMap`). *A* `ContainerMap` *is a tuple* $\langle u_{\text{cm}}, \mathbf{RM}, \mathbf{CM}, \mathbf{NM} \rangle$ *where* u_{cm} *is an IRI,* $\mathbf{RM}$ *is a set of* `ResourceMaps`*,* $\mathbf{CM}$ *is a set of* `ContainerMaps`*, and* $\mathbf{NM}$ *is a set of* `NonContainerMaps`*.*

Consider the `ContainerMap` `:dataset` in Figure 4.6. Supposing that its `ResourceMap` `:rm2` is described by rm_2, its `NonContainerMap` `:publisher` is described by $publisher$ and its `ContainerMaps` `:distribs` and `:themes` are described by $distribs$ and $themes$ respectively. Then, in the abstract syntax, the `ContainerMap` `:dataset` is described by $dataset = \langle$`:distribs`$, rm_2, \{distribs, themes\}, \{publisher\}\rangle$.

Design Document

A design document contains a set of `ContainerMaps` and `NonContainerMaps`. Its evaluation starts from those `ContainerMaps` and `NonContainerMaps` are not contained in any `ContainerMap`.

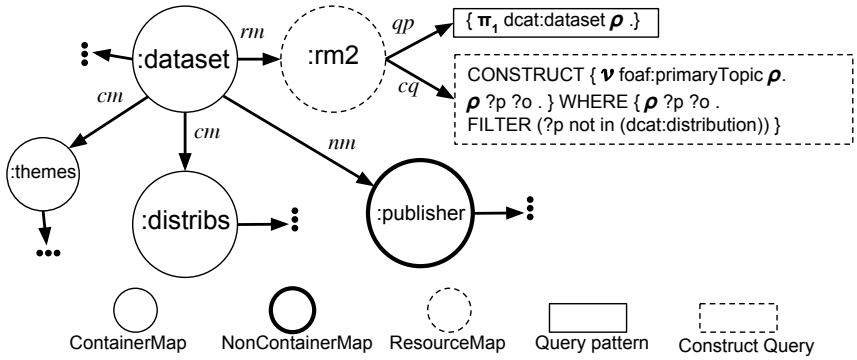

FIGURE 4.6: Example of a NonContainerMap

Definition 5 (Design Document). *A* design document *in LDP-DL is a pair* $\langle \mathbf{CM}, \mathbf{NM} \rangle$, *where* $\mathbf{CM}$ *is a set of* ContainerMaps *and* $\mathbf{NM}$ *is a set of* NonContainerMaps. *Only* ContainerMaps *and* NonContainerMaps *that do not have a parent* ContainerMap *are in* $\mathbf{CM}$ *and* $\mathbf{NM}$ *respectively.*

For example, the design document in Figure 4.2 can be described by $\langle \{catalog\}, \{\} \rangle$ in the abstract syntax. As we can see, only $catalog$ in the set of ContainerMaps of the design document as it does not have any parent ContainerMap. The evaluation of this design document starts with $catalog$ and then recursively evaluates the child ContainerMaps and NonContainerMaps.

Let us now proceed with the semantics of LDP-DL in the next section.

4.2.3 Model-theoretic Semantics

In this section, we provide the semantics of LDP-DL in a model-theoretic way using a notion of interpretation and a notion of satisfaction. To this end, in Section 4.2.3 and Section 4.2.3, we describe the notion of interpretation and satisfaction respectively.

Notation Before proceeding, we provide some notations that we use in our formalization below.

IRI of LDP-DL Construct Given a ContainerMap, NonContainerMap, ResourceMap or DataSource x, we refer to the IRI of x as iri(x).

Ancestor List A list of ancestors is a finite sequence of elements that can be IRIs or a special value $\epsilon \notin \mathbf{IRI}$ that indicates an absence of a resource. Formally, an *ancestor list* is an element of $\mathbf{IRI}^* = \bigcup_{n>=0} (\mathbf{IRI} \cup \{\epsilon\})^n$ and $\emptyset$ being the empty list $(\mathbf{IRI} \cup \{\epsilon\})^0$. We use the notation $\vec{p}$ to denote an ancestor list and use $\text{len}(\vec{p})$ to denote the length of the list. Also $\vec{p} :: r$ denotes appending element r to $\vec{p}$.

LDP-DL interpretation

An LDP-DL interpretation determines which IRIs denote ContainerMaps, NonContainerMaps, ResourceMaps, DataSources, or something else. Then, each ContainerMap and NonContainerMap is interpreted as a set of triples $(url, graph, M)$, representing

containers, and a set of pairs $(url, graph)$, representing non-containers, respectively with respect to a list of ancestors.

Definition 6 (LDP-DL Interpretation). *An* LDP-DL interpretation $\mathcal{I}$ *is a tuple* $\langle \Delta^{\mathcal{I}}, \mathcal{C}, \mathcal{N}, \mathcal{R}, \mathcal{S}, \cdot^{\mathcal{I}}, \mathcal{I}_{\mathcal{C}}, \mathcal{I}_{\mathcal{N}}, \mathcal{I}_{\mathcal{R}}, \mathcal{I}_{\mathcal{S}} \rangle$ *such that:*

- $\Delta^{\mathcal{I}}$ *is a non empty set (the* domain *of interpretation);*
- $\mathcal{C}, \mathcal{N}, \mathcal{R}, \mathcal{S}$ *are subsets of* $\Delta^{\mathcal{I}}$*;*
- $\cdot^{\mathcal{I}} : \mathbf{IRI} \to \Delta^{\mathcal{I}}$ *is the* interpretation function*;*
- $\mathcal{I}_{\mathcal{C}} : \mathcal{C} \times \mathbf{IRI}^* \to 2^{\mathbf{IRI} \times \mathcal{G} \times 2^{\mathbf{IRI}}}$*;*
- $\mathcal{I}_{\mathcal{N}} : \mathcal{N} \times \mathbf{IRI}^* \to 2^{\mathbf{IRI} \times \mathcal{G}}$*;*
- $\mathcal{I}_{\mathcal{R}} : \mathcal{R} \times \mathbf{IRI}^* \to 2^{\mathbf{IRI} \times \mathbf{IRI} \cup \{\epsilon\} \times \mathcal{G}}$ *such that* $(n, r_1, g_1) \in \mathcal{I}_{\mathcal{R}}(u_1, \vec{p_1}) \wedge (n, r_2, g_2) \in \mathcal{I}_{\mathcal{R}}(u_2, \vec{p_2}) \implies r_1 = r_2 \wedge g_1 = g_2$ *(`unicity constraint`);*
- $\mathcal{I}_{\mathcal{S}} : \mathcal{S} \to \mathcal{G}$*.*

We describe the different parts of an LDP-DL interpretation below.

$\Delta^{\mathcal{I}}$ & Interpretation function: $\Delta^{\mathcal{I}}$ is a non-empty set that is the domain of interpretation. Elements of $\Delta^{\mathcal{I}}$ are accessed using their IRIs through the interpretation function $\cdot^{\mathcal{I}}$.

$\mathcal{C}, \mathcal{N}, \mathcal{R}, \mathcal{S}$: $\mathcal{C}, \mathcal{N}, \mathcal{R}$ and $\mathcal{S}$ represent the container maps, non-container maps, resource maps and data sources according to the interpretation. That is, if the interpretation function $\mathcal{I}$ maps an IRI to an element of $\mathcal{C}$, it means that this interpretation considers that the IRI is the name of a container map. The same follows for elements of $\mathcal{N}$, $\mathcal{R}$ and $\mathcal{S}$.

$\mathcal{I}_{\mathcal{C}} : \mathcal{C} \times \mathbf{IRI}^* \to 2^{\mathbf{IRI} \times \mathcal{G} \times 2^{\mathbf{IRI}}}$: For a given `ContainerMap` $cm \in \mathcal{C}$ and an ancestor list $\vec{p}$, $\langle n, g, M \rangle \in \mathcal{I}_{\mathcal{C}}(cm, \vec{p})$ means that, in the context of $\vec{p}$, cm must map to containers where n is the IRI of a container, g is the RDF graph obtained from dereferencing n, and M is the set of IRIs referring to the members of the container.

$\mathcal{I}_{\mathcal{N}} : \mathcal{N} \times \mathbf{IRI}^* \to 2^{\mathbf{IRI} \times \mathcal{G}}$: For a given `NonContainerMap` $nm \in \mathcal{N}$, $\langle n, g \rangle \in \mathcal{I}_{\mathcal{N}}(nm, \vec{p})$ means that, in the context of $\vec{p}$, nm must map to non-containers where n is the IRI of a non-container and g is the RDF graph obtained from dereferencing n.

$\mathcal{I}_{\mathcal{R}} : \mathcal{R} \times \mathbf{IRI}^* \to 2^{\mathbf{IRI} \times \mathbf{IRI} \cup \{\epsilon\} \times \mathcal{G}}$: For a given `ResourceMap` $rm \in \mathcal{R}$, $(n, r, g) \in \mathcal{I}_{\mathcal{R}}(rm, \vec{p})$ means that rm must map to a triple where n is the IRI of a new LDP resource, r is the related resource of the new LDP resource and g is the RDF graph obtained from dereferencing n. The `unicity constraint` guarantees that LDP resources from the interpretation of a `ResourceMap` are unique.

$\mathcal{I}_\mathcal{S} : \mathcal{S} \to \mathcal{G}$ For a given `DataSource` $ds \in \mathcal{S}$, $\mathcal{I}_\mathcal{S}(ds)$ is an RDF graph obtained from the data source. Here, we abstract from the different forms that ds can take and defines its interpretation at a higher level.

Satisfaction

This section formally explains the semantics of LDP-DL by describing the satisfaction $\models$ of its constructs with respect to an LDP-DL interpretation. In the rest of this section, we do so in a bottom-up approach starting with the satisfaction of a `DataSource`.

Satisfaction of a `DataSource` In principle, a `DataSource` can take several forms to provide information for retrieving an RDF graph. As mentioned before, we define only two forms of `DataSources`, $ds = \langle u_{\mathsf{ds}}, u_{\mathsf{loc}} \rangle$ that provides the URL u_{loc} of RDF document directly, and $ds = \langle u_{\mathsf{ds}}, u_{\mathsf{loc}}, u_{\mathsf{lr}} \rangle$ that provides the URL u_{loc} of an arbitrary document and the URL u_{lr} of a transformation script to generate an RDF graph. A `DataSource` ds is satisfied if, by using the parameters of ds, we obtain the RDF graph of the interpretation of $\mathsf{iri}(ds)$.

Formally, the satisfaction of `DataSources` is parameterized by the dereference function deref and the lifting rule map lr. Assume the existence of an infinite set $\mathcal{D}$ whose elements are *documents*. The dereference function $\mathsf{deref} : \mathbf{IRI} \to \mathcal{D}$ assigns a document to an IRI. A lifting rule $\mathsf{lif} : \mathcal{D} \to \mathcal{G}$ is a function that assigns an RDF graph to a document. Finally, the lifting rule Map $\mathsf{lr} : \mathbf{IRI} \nrightarrow (\mathcal{D} \to \mathcal{G})$ is a partial function assigning a lifting rule to some IRIs.

With respect to deref and lr, the satisfaction of a `DataSource` is defined below.

Definition 7. *Let $\mathcal{I}$ be a LDP-DL interpretation. We say that $\mathcal{I}$* satisfies *$ds = \langle u_{\mathsf{ds}}, u_{\mathsf{loc}}, u_{\mathsf{lr}} \rangle$ with respect to a function* deref *and a lifting rule map* lr*, written $\mathcal{I} \models_{\mathsf{deref},\mathsf{lr}} ds$ iff $\mathcal{I}_\mathcal{S}(u_{\mathsf{ds}}^\mathcal{I}) = \mathsf{lr}(\mathsf{deref}(u_{\mathsf{lr}}))(\mathsf{deref}(u_{\mathsf{loc}}))$. Similarly, if $ds = \langle u_{\mathsf{ds}}, u_{\mathsf{loc}} \rangle$, then $\mathcal{I}$ satisfies ds with respect to* deref *iff $\mathcal{I}_\mathcal{S}(u_{\mathsf{ds}}^\mathcal{I}) = \mathsf{deref}(u_{\mathsf{loc}})$. If $\mathbf{DS}$ is a set of `DataSources`, then we write $\mathcal{I} \models \mathbf{DS}$ when for all $ds \in \mathbf{DS}$, $\mathcal{I} \models ds$.*

As we can see, our semantics consider only two forms of `DataSources` but can extended to more complex such as for access rights, content negotiation etc. When there is no ambiguity or no need to specify deref and lr, we simply write $\mathcal{I} \models ds$.

Let us consider an example of a `DataSource` $ds_1 = \langle \texttt{ex:source1}, \texttt{ex:graph.ttl} \rangle$ that describes a particular RDF data source. The interpretation of the data source $\mathcal{I}_\mathcal{S}(\texttt{ex:source1}^\mathcal{I})$ may be the RDF graph in Figure 4.7.

Satisfaction of a `ResourceMap` The satisfaction of a `ResourceMap` $\langle u_{\mathsf{rm}}, \mathsf{qp}, \mathsf{cq}, \mathbf{DS} \rangle$ is defined with respect to an ancestor list, and depends on the results of the query evaluation of qp and cq over the RDF graph obtained by doing an RDF merge on all the RDF graphs obtained from $\mathbf{DS}$. To define this properly, we need to introduce additional notations.

For a given ancestor list $\vec{p}$, let $\mu_{\vec{p}}$ be the mapping s.t. $\mathrm{dom}(\mu_{\vec{p}}) \subseteq \{\pi_i \mid 1 \leq i \leq \mathrm{len}(\vec{p})\}$ and for $1 \leq i \leq \mathrm{len}(\vec{p})$, $\mu_{\vec{p}}(\pi_i) = \vec{p}[i]$ if $\vec{p}[i] \neq \epsilon$ and $\mu_{\vec{p}}(\pi_i)$ undefined otherwise. This mapping can be used to constraints the selection of related resources and the construction of the associated graphs. We call $\mu_{\vec{p}}$ the *ancestor mapping* and the variables π_i the *ancestor variables*.

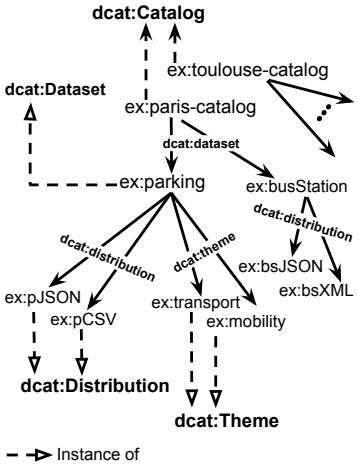

FIGURE 4.7: Example of an interpretation of a data source

Definition 8. *Let $\mathcal{I}$ be an LDP-DL interpretation, $\vec{p}$ an ancestor list, $rm = \langle u_{\mathsf{rm}}, \mathsf{qp}, \mathsf{cq}, \mathbf{DS}\rangle$ be a* `ResourceMap`*, with* $\mathsf{cq} = \langle \mathsf{gt}_{\mathsf{cq}}, \mathsf{qp}_{\mathsf{cq}}\rangle$*. Let* $g_s = \mathrm{Merge}(\{\mathcal{I}_{\mathcal{S}}(\mathsf{iri}(ds)^{\mathcal{I}}) \mid ds \in \mathbf{DS}\})$*. We say that $\mathcal{I}$* satisfies *rm with respect to $\vec{p}$, written $\mathcal{I}, \vec{p} \models rm$, iff:*

- $\mathcal{I} \models \mathbf{DS}$*;*
- *if* $r \in \Pi_\rho([\![\mathsf{qp}]\!]_{g_s} \bowtie \{\mu_{\vec{p}}\})$ *then there exists* $\langle n, r, g\rangle \in \mathcal{I}_{\mathcal{R}}(u^{\mathcal{I}}_{\mathsf{rm}}, \vec{p})$*;*
- *if* $\langle n, r, g\rangle \in \mathcal{I}_{\mathcal{R}}(u^{\mathcal{I}}_{\mathsf{rm}}, \vec{p})$ *then:*
  - $r \in \Pi_\rho([\![\mathsf{qp}]\!]_{g_s} \bowtie \{\mu_{\vec{p}}\})$,
  - $g \supseteq \mathsf{gt}_{\mathsf{cq}}([\![\mathsf{qp}_{\mathsf{cq}}]\!]_{g_s}) \bowtie \{\mu_\nu\} \bowtie \{\mu_\rho\} \bowtie \{\mu_{\vec{p}}\}$*;*

  where:

  - *let μ_ν be the mapping where* $\mathrm{dom}(\mu_\nu) = \{\nu\}$ *and* $\mu_\nu(\nu) = n$*,*
  - *let μ_ρ be the mapping where* $\mathrm{dom}(\mu_\rho) = \{\rho\}$ *and* $\mu_\rho(\rho) = r$ *if* $r \neq \epsilon$ *and $\mu_\rho(\rho)$ undefined otherwise.*

If $\mathbf{RM}$ *is a set of* `ResourceMap`*s, then we write* $\mathcal{I}, \vec{p} \models \mathbf{RM}$ *when for all* $rm \in \mathbf{RM}$*,* $\mathcal{I}, \vec{p} \models rm$*.*

Let us now describe the conditions in the definition using the `ResourceMap` `:rm2` in Figure 4.8 (a).

Suppose that the interpretation of `:rm2` with respect to the ancestor list (`ex:paris-catalog`), shown in Figure 4.8 (b), returns the triples $\langle$`dex:parking`, `ex:parking`, $g_1\rangle$ and $\langle$`dex:busStation`, `ex:busStation`, $g_2\rangle$.

First Condition This condition ensures that `DataSources` of a `ResourceMap` are well interpreted. In our example, we assume this condition is satisfied with the RDF graph in Figure 4.7 being the interpretation of the `DataSource` of `:rm2`.

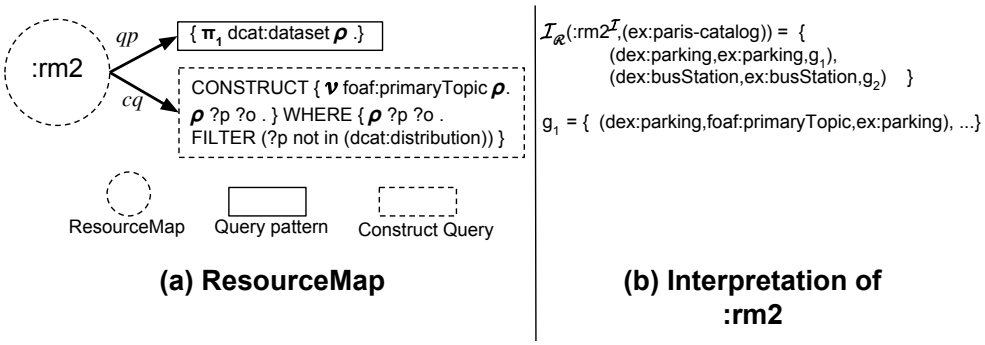

FIGURE 4.8: Example of a ResourceMap and an RDF Graph as its DataSource

Second Condition This condition ensures that for all related resources extracted, a triple $\langle n, r, g\rangle$ is generated that as we describe in the definitions below, is used to generate containers and non-containers. The evaluation of the query pattern of :rm2 with π_1 bounded to ex:paris-catalog returns the related resource ex:parking and ex:busStation. This condition is satisfied since for both of them, a triple exist in the interpretation of :rm2.

Third Condition This condition places constraints on the triples in the interpretation of a ResourceMap. In the case of :rm2, the related resources ex:parking and ex:busStation in the triples in its interpretation are obtained from the evaluation of its query pattern. As we can see, we require the graph of the LDP resource be a superset of the evaluation of cq to deal with the openness of the LDP standard related to the content of RDF sources. For example, [SAM15a, §5.2.1.1] states that their content may not obligatorily include a triple to type them as an ldp:RDFSource. Thus, assuming that both a subset of g_1 and g_2 were obtained from the evaluation of the CONSTRUCT query of :rm2, the interpretation of :rm2 satisfies this condition.

Satisfaction of a NonContainerMap An interpretation satisfies a NonContainerMap $\langle u_{\mathsf{nm}}, \mathbf{RM}\rangle$ with respect to an ancestor list when it satisfies all $rm \in \mathbf{RM}$, and the set of named graphs associated with u_{nm} tallies with the interpretations of the ResourceMaps.

Definition 9. *Let $\mathcal{I}$ be a LDP-DL interpretation, $\vec{p}$ an ancestor list, $nm = \langle u_{\mathsf{nm}}, \mathbf{RM}\rangle$ be a* NonContainerMap*.*

We say that $\mathcal{I}$ satisfies *nm with respect to $\vec{p}$, written $\mathcal{I}, \vec{p} \models nm$, iff:*

- $\mathcal{I}, \vec{p} \models \mathbf{RM}$*;*
- *if $\langle n, g\rangle \in \mathcal{I}_{\mathcal{N}}(u^{\mathcal{I}}_{\mathsf{nm}}, \vec{p})$ then there exists $rm \in \mathbf{RM}$, $\langle n, r, g'\rangle \in \mathcal{I}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}}, \vec{p})$ such that $g' \subseteq g$;*
- *if there exists $rm \in \mathbf{RM}$, $\langle n, r, g'\rangle \in \mathcal{I}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}}, \vec{p})$, then there exists a unique $\langle n, g\rangle \in \mathcal{I}_{\mathcal{N}}(u^{\mathcal{I}}_{\mathsf{nm}}, \vec{p})$ such that $g' \subseteq g$.*

If $\mathbf{NM}$ is a set of NonContainerMap*, then we write $\mathcal{I}, \vec{p} \models \mathbf{NM}$ when for all $nm \in \mathbf{NM}$, $\mathcal{I}, \vec{p} \models nm$.*

Let us now describe the three conditions in the above definition using the NonContainerMap :distrib in Figure 4.9 (a). Let's assume that $\langle$dex:pJSON, $g_1\rangle$ and $\langle$dex:pCSV, $g_2\rangle$ are in the interpretation of :distrib with the ancestor list (ex:paris-catalog,ex:parking,ϵ) shown in Figure 4.9 (b)

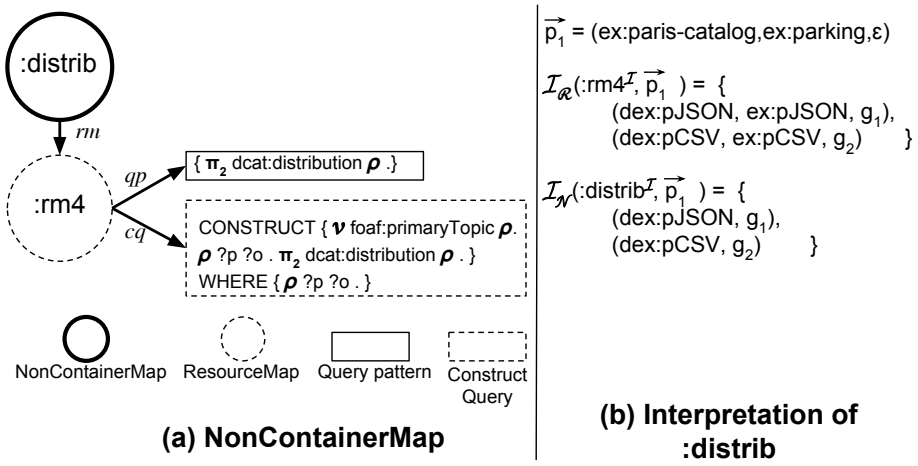

FIGURE 4.9: Example of a NonContainerMap

First Condition This condition ensures all ResourceMaps of a NonContainerMap are well satisfied. In our example, let us assume that this condition is satisfied with the triple $\langle$dex:pJSON, ex:pJSON, $g_1\rangle$ and $\langle$dex:pCSV, ex:pCSV, $g_2\rangle$ being in the interpretation of :rm4 with the ancestor list (ex:paris-catalog,ex:parking,ϵ).

Second Condition This conditions ensures that a non-container $\langle n, g\rangle$ in the interpretation of a NonContainerMap nm exist with respect to a triple $\langle n, r, g'\rangle$ in the interpretation of nm's ResourceMap with $g' \subseteq g$. In our example, $\langle$dex:pJSON, $g_1\rangle$ and $\langle$dex:pCSV, $g_2\rangle$ in the interpretation of :distrib exist with respect to $\langle$dex:pJSON, ex:pJSON, $g_1\rangle$ and $\langle$dex:pCSV, ex:pCSV, $g_2\rangle$ in the interpretation of :rm4. The graph of non-containers (i.e. g_1, g_2) are the same graph from those in the interpretation of :rm4 and therefore this condition is satisfied in our example.

Third Condition This condition ensures that for every triple $\langle n, r, g'\rangle$ in the interpretation of a ResourceMap rm, there is a unique non-container $\langle n, g\rangle$ in the interpretation of rm's NonContainerMap. In our example, $\langle$dex:pJSON, ex:pJSON, $g_1\rangle$ and $\langle$dex:pCSV, ex:pCSV, $g_2\rangle$ are in the interpretation of :rm4 and $\langle$dex:pJSON, $g_1\rangle$ and $\langle$dex:pCSV, $g_2\rangle$ are in the interpretation of :distrib. Also, The graph of non-containers (i.e. g_1, g_2) are the same graph from those in the interpretation of :rm4 and therefore this condition is satisfied in our example.

Satisfaction of a ContainerMap As mentioned previously, the satisfaction of a ContainerMap $\langle u_{\text{cm}}, \mathbf{RM}, \mathbf{CM}, \mathbf{NM}\rangle$ is defined with respect to an ancestor list. Constraints similar to those of NonContainerMaps apply. In addition, the maps in $\mathbf{CM}$ and $\mathbf{NM}$ must be satisfied with respect to a new ancestor list that is the previous list appended with the related resources of the ContainerMap.

Additionally, to satisfy a `ContainerMap`, the set of members M in a triple $\langle n, g, M\rangle$ must contain the resources that must be generated by the maps of $\mathbf{CM}$ and $\mathbf{NM}$ with respect to the new ancestor list.

Definition 10. *Let $\mathcal{I}$ be a LDP-DL interpretation, $\vec{p}$ an ancestor list, $cm = \langle u_{\mathsf{cm}}, \mathbf{RM}, \mathbf{CM}, \mathbf{NM}\rangle$ be a* `ContainerMap`.

We say that $\mathcal{I}$ satisfies *cm with respect to $\vec{p}$, written $\mathcal{I}, \vec{p} \models nm$, iff:*

- $\mathcal{I}, \vec{p} \models \mathbf{RM}$;
- *if $\langle n, g, M\rangle \in \mathcal{I}_{\mathcal{C}}(u^{\mathcal{I}}_{\mathsf{cm}}, \vec{p})$ then there exists $rm \in \mathbf{RM}$, $\langle n, r, g'\rangle \in \mathcal{I}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}}, \vec{p})$ such that $g' \subseteq g$;*
- *if there exists $rm \in \mathbf{RM}$, $\langle n, r, g'\rangle \in \mathcal{I}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}}, \vec{p})$, then there exists a unique $\langle n, g, M\rangle \in \mathcal{I}_{\mathcal{C}}(u^{\mathcal{I}}_{\mathsf{cm}})$ such that $g' \subseteq g$;*
- *for all $rm \in \mathbf{RM}$ and for all $\langle n, r, g\rangle \in \mathcal{I}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}}, \vec{p})$ and for all $\langle n, g_n, M_n\rangle \in \mathcal{I}_{\mathcal{C}}(u^{\mathcal{I}}_{\mathsf{cm}}, \vec{p})$*
  - $\mathcal{I}, \vec{p} :: r \models \mathbf{NM}$ *and* $M_n \supseteq \{n \mid \exists nm \in \mathbf{NM} \wedge \exists \langle n, g\rangle \in \mathcal{I}_{\mathcal{N}}(\mathsf{iri}(nm)^{\mathcal{I}}, \vec{p} :: r)\}$;
  - $\mathcal{I}, \vec{p} :: r \models \mathbf{CM}$ *and* $M_n \supseteq \{n \mid \exists cm \in \mathbf{CM} \wedge \exists \langle n, g, M'\rangle \in \mathcal{I}_{\mathcal{C}}(\mathsf{iri}(cm)^{\mathcal{I}}, \vec{p} :: r)\}$.
  - $n' \in M_n \Rightarrow (\exists nm \in \mathbf{NM}, \exists g', \langle n', g'\rangle \in \mathcal{I}_{\mathcal{N}}(\mathsf{iri}(nm)^{\mathcal{I}}, \vec{p} :: r)) \vee (\exists cm' \in \mathbf{CM}, \exists g', \exists M', \langle n', g', M'\rangle \in \mathcal{I}_{\mathcal{C}}(\mathsf{iri}(cm')^{\mathcal{I}}, \vec{p} :: r)$

If $\mathbf{CM}$ is a set of `ContainerMap`*, then we write $\mathcal{I}, \vec{p} \models \mathbf{CM}$ when for all $cm \in \mathbf{CM}$, $\mathcal{I}, \vec{p} \models cm$.*

Let us now describe the four main conditions in the above definition using the `ContainerMap` `:distribs` in Figure 4.10 (a). Let's assume that $\langle$`dex:pDistributions`$, g_1, M_1\rangle$ with M_1 ={`dex:pJSON`,`dex:pCSV`} is in the interpretation of `:distribs`, shown in Figure 4.10 (b), with the ancestor list (`ex:paris-catalog`,`ex:parking`).

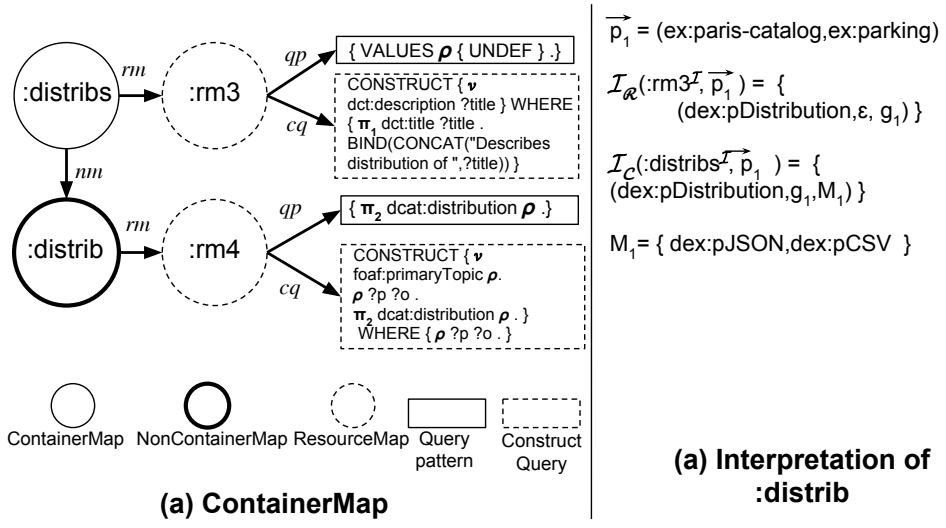

FIGURE 4.10: Example of a `ContainerMap`

First Condition The first condition ensures that ResourceMaps are well satisfied. In our example, we assume that this condition is satisfied with the RDF graph in Figure 4.7 being the interpretation of the DataSource of :rm3 and, $\langle$dex:pDistributions, ϵ, $g_1\rangle$ is the interpretation of :rm3.

Second Condition This conditions ensures that a container $\langle n, g, M\rangle$ in the interpretation of a NonContainerMap cm exist with respect to a triple $\langle n, r, g'\rangle$ in the interpretation of cm's ResourceMap with $g' \subseteq g$. In our example, $\langle$dex:pDistributions, g_1, $M_1\rangle$ is in the interpretation of :distribs with respect to $\langle$dex:pDistributions, ϵ, $g_1\rangle$ being in the interpretation of :rm3. The graph g_1 of dex:pDistributions is the same graph as in the interpretation of :rm3 and therefore this condition is satisfied in our example.

Third Condition This condition ensures that for every triple $\langle n, r, g'\rangle$ in the interpretation of a ResourceMap rm, there is a unique container $\langle n, g, M\rangle$ in the interpretation of rm's ContainerMap. In our example, $\langle$dex:pDistributions, ϵ, $g_1\rangle$ are in the interpretation of :rm3 and $\langle$dex:pDistributions, g_1, $M_1\rangle$ is in the interpretation of :distribs. Also, The graph g_1 of dex:pDistributions is the same graph as in the interpretation of :rm3 and therefore this condition is satisfied in our example.

Fourth Condition In short, this condition only ensures that a container $\langle n, g, M\rangle$ in the interpretation of a ContainerMap cm contains the proper members. By proper members, we refer to containers and non-containers in the interpretation of the ContainerMap and NonContainerMap of cm respectively. In the example of Definition 9, we explained the satisfaction of the NonContainerMap :distrib. We also mentioned that its interpretation with the ancestor list (ex:paris-catalog,ex:parking,ϵ) contains the non-containers dex:pJSON dex:pCSV. :distribs contains only the NonContainerMap :distrib and no other ContainerMaps. This is why dex:pDistributions contains only dex:pJSON dex:pCSV as members.

Satisfaction of an LDP-DL document An LDP-DL document is satisfied if all its top level ContainerMaps and NonContainerMaps are satisfied with respect to an empty ancestor list.

Definition 11. *Let $\mathcal{I}$ be an LDP-DL interpretation, $\delta = \langle \mathbf{CM}, \mathbf{NM}\rangle$ be an LDP-DL document.*

We say that $\mathcal{I}$ satisfies *δ , written $\mathcal{I} \models \delta$, iff*

- $\mathcal{I}, \emptyset \models \mathbf{CM}$ *and*
- $\mathcal{I}, \emptyset \models \mathbf{NM}$.

The interpretation of a design document starts with the interpretation of its $\mathbf{CM}$ and $\mathbf{NM}$ with respect to an empty ancestor list. The $\mathbf{CM}$ and $\mathbf{NM}$ contains ContainerMaps and NonContainerMaps from the design document that have no parent ContainerMap. Consider the LDP-DL design in Figure C.1. In its design document, $\mathbf{CM}$ would contain only :catalog and $\mathbf{NM}$ would be empty.

Now that we have defined the satisfaction of an LDP-DL document with respect to an interpretation, let us proceed with defining the evaluation of a design document.

4.3 LDP Dataset

An LDP dataset is a model to store resources in an LDP. In this section, in Section 4.3.1, we describe the evaluation of a design document to generate an LDP dataset. Then, in Section 4.3.2, we describe variabilities that can affect the validity of an LDP dataset and provide ways to abstract them.

4.3.1 Design Document Evaluation

With an interpretation, we have a way of assigning an LDP dataset to a design document, using the interpretations of the `ContainerMaps` and `NonContainerMaps` that appear in the document. We call this an *evaluation* of the document. Formally, it takes the form of a function that builds an LDP dataset given an LDP-DL interpretation and a document δ. We formalize the notion of LDP dataset as follows:

Definition 12 (LDP dataset). *An* LDP dataset *is a pair* $\langle \mathbf{NG}, \mathbf{NC} \rangle$ *where* $\mathbf{NG}$ *is a set of named graphs and* $\mathbf{NC}$ *is a set of* named container*, that is a set of triples* $\langle n, g, M \rangle$ *such that* $n \in \mathbf{IRI}$ *(called the container name),* $g \in \mathcal{G}$ *and* $M \in 2^{\mathbf{IRI}}$*. In addition to this, there are some constraints on an LDP dataset. In an LDP dataset* Σ*:*

- *no IRI appears more than once as a (graph or container) name;*
- *for all* $\langle n, g, M \rangle \in \mathbf{NC}$*, and for all* $u \in M$*, there exists a named graph or container having the name* u *in* Σ*.*

Having the notion of LDP dataset, the maps of the design document is evaluated with respect to a ancestor list as follows:

Definition 13 (Evaluation of a map). *The* evaluation *of a* `ContainerMap` *or* `NonContainerMap` m *with respect to an interpretation* $\mathcal{I}$ *and an ancestor list* $\vec{p}$ *s.t.* $\mathcal{I}, \vec{p} \models m$ *is:*

$$[\![m]\!]^{\vec{p}}_{\mathcal{I}} = \begin{cases} \mathcal{I}_{\mathcal{N}}(\mathsf{iri}(m)^{\mathcal{I}}, \vec{p}), & \text{if } m \text{ is a } \texttt{NonContainerMap} \\ \mathcal{I}_{\mathcal{C}}(\mathsf{iri}(m)^{\mathcal{I}}, \vec{p}) \cup \bigcup\limits_{\substack{rm \in \mathbf{RM} \\ \langle n,r,g \rangle \in \mathcal{I}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}}, \vec{p}) \\ m' \in \mathbf{NM} \cup \mathbf{CM}}} [\![m']\!]^{\vec{p}::r}_{\mathcal{I}}, & \text{if } m = \langle u_{\mathsf{cm}}, \mathbf{RM}, \mathbf{CM}, \mathbf{NM} \rangle \text{ is a } \texttt{ContainerMap} \end{cases}$$

The evaluation of a map yields an LDP dataset. Indeed, the first condition of Definition 12 is satisfied because of the unicity constraint from Definition 6, and the second condition is satisfied because $\mathcal{I}, \vec{p} \models m$. Now we can define the evaluation of a design document with respect to an interpretation:

Definition 14 (Evaluation of an design document). *Let $\mathcal{I}$ be an interpretation and $\delta = \langle \mathbf{CM}, \mathbf{NM} \rangle$ a design document. The* evaluation *of δ with respect to $\mathcal{I}$ is*

$$[\![\delta]\!]_{\mathcal{I}} = \bigcup_{m \in \mathbf{CM} \cup \mathbf{NM}} [\![m]\!]^{\emptyset}_{\mathcal{I}}$$

In practice, an LDP-DL processor will not define an explicit interpretation, but will build an LDP dataset from a design document. Hence, we want to define a notion of conformity of an algorithm with respect to the language specification given above. To this aim, we first provide a definition of a valid LDP dataset for a design document.

Definition 15 (Valid). *An LDP dataset D is* valid *with respect to a design document δ if there exists an interpretation $\mathcal{I}$ that satisfies δ, such that $[\![\delta]\!]_{\mathcal{I}} = D$.*

Finally, we can define the correctness of an algorithm that implements LDP-DL.

Definition 16 (Correct). *An algorithm that takes an LDP-DL document as input and returns an LDP dataset is correct if for all document δ, the output of the algorithm on δ is valid with respect to δ.*

4.3.2 Variability Abstraction

After that a design document has been evaluated to obtain an LDP dataset, due to some variabilities, the LDP dataset can be invalidated. In this section, we describe these variabilities in Section 4.3.2. Then, we propose two models namely static and dynamic LDP in Section 4.3.2 and Section 4.3.2 respectively to abstract these variabilities.

Variabilities of LDP Dataset

Two aspects whose variability can invalidate an LDP dataset is the LDP server location and dynamicity of data sources. Below, we discuss them in more details.

LDP Server Location After that an LDP dataset has been generated, a change in the LDP server location and/or data sources may invalidate the LDP dataset. In the current definition of LDP datasets (Definition 12), LDP resources are identified using their IRIs. The problem with hardcoding IRIs in the LDP dataset is that the server where the LDP will be deployed has to be decided prior to generating the LDP dataset. To some extent, this make the deployment dependent on the design. In Section 4.3.2, we explain how we abstract this change using *static LDP datasets*.

Dynamicity of Data Sources To explain the dynamicity of data sources, let us suppose that t_x and t_y are two time instants such that $t_y > t_x$. With respect to a design document δ, the LDP dataset generated at t_x may not be valid at t_y due to changes that may have occurred at data sources referred in δ between t_x and t_y. Below, we identify two types of changes at these data sources whose occurrence may invalidate the LDP dataset generated at t_x.

Change of Resources The first change, that we refer to *change of resources*, occurs when the related resources obtained from the evaluation of the query patterns of any `ResourceMaps` in δ at t_y is different from those at t_x. Consequently, this invalidates the LDP dataset as it may contain LDP resources describing related resources already deleted at a data source or it may not contain LDP resources for describing related resource newly added at a data source.

Consider the LDP-DL design in Figure 4.2. Suppose that the query pattern of `:rm1` returns a new resource that was not returned at time t_x. For that resource, a new LDP resource need to be created. In addition to this, with respect to that new resource, a number of new LDP resources may need be created from the remaining `ContainerMaps` and `NonContainerMaps` such as `:dataset`, `:distribs`, etc.

Dealing with this change may be both complex and resource intensive. An optimal approach would be to identify the specific part of the design document that need to be re-evaluated when a change occurs. In this thesis, we do not deal specifically address this change but it is possible to naively address it by evaluating the complete design document every time a change occurs at the data source or whenever a request for an LDP resource received.

Change in Resources' Descriptions The second change, that we refer to as *change in resources' descriptions*, occurs when only the evaluation results of the CONSTRUCT query of any `ResourceMaps` in δ at t_y are different from their evaluation results at t_x. Consequently, this invalidates the LDP dataset as the content of LDP resources is valid with respect to the design document and data source.

To illustrate this change in the LDP-DL design in Figure 4.2, if the evaluation results of the CONSTRUCT of `:rm1` for a particular LDP resource is different at t_y, then only the content of that LDP resource in invalid. Compared to the previous change, this change is isolated only to specific LDP resources making it easier to deal with. In Section 4.3.2, we explain the use of *dynamic LDP datasets* to abstract this change.

Static LDP Dataset

To handle the variability of the location of LDP servers, we use *static LDP datasets* to abstract from the location of servers where LDPs will be deployed. A static LDP dataset is a data structure similar to an LDP dataset except that rather than using IRIs as names for LDP resources, temporary identifiers are used. Thus, as shown in Figure 4.11, a particular static LDP dataset may be generated and used in several deployment scenarios. More precisely, during deployment, an LDP dataset is generated from the static LDP dataset by by replacing the temporary identifiers with final IRIs for LDP resources based on the location of the server.

Before formally defining static LDP dataset, we define $\mathbf{ID}$ as the set of unique identifiers that may include letters, digits, words,numbers, etc. We define this here because we exclusively use it for static and dynamic LDP datasets (described in the next section).

Definition 17 (Static LDP dataset). *An* Static LDP dataset *is a pair* $\langle \mathbf{NSG}, \mathbf{NSC} \rangle$. $\mathbf{NSG}$ *is a set of static named graphs* $\langle id, g \rangle$ *where* $id \in \mathbf{ID}$ *and* $g \in \mathcal{G}$. $\mathbf{NSC}$ *is a set*

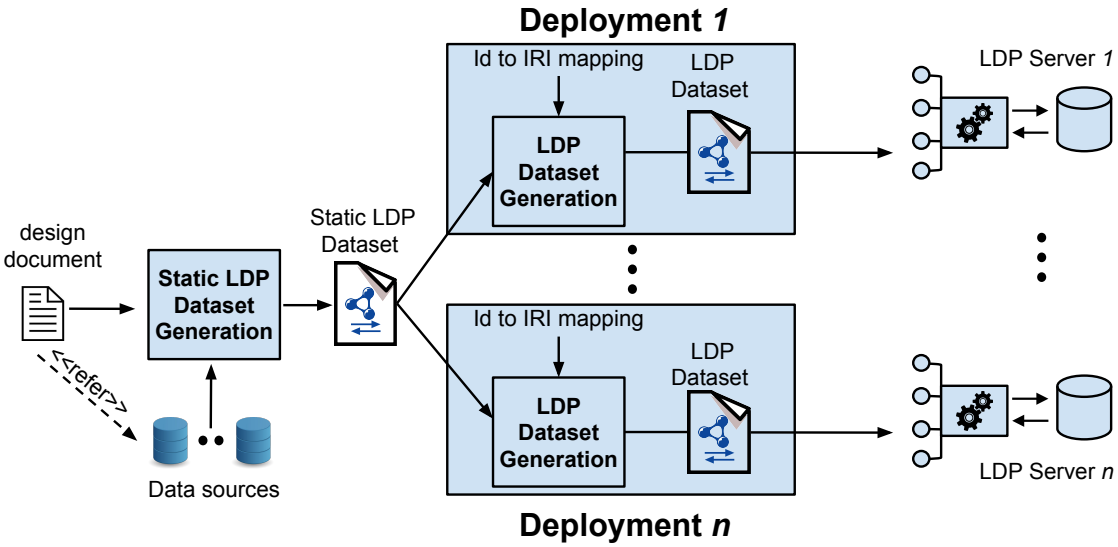

FIGURE 4.11: Application of static LDP datasets

of static named container $\langle id, g, M\rangle$ *such that* $id \in \mathbf{ID}$ *(called the static container name),* $g \in \mathcal{G}$ *and* $M \in 2^{\mathbf{ID}}$*. In addition to this, there are some constraints on a static LDP dataset. In an LDP dataset* Σ*:*

- *no* id *appears more than once as a static named graph or container name;*
- *for all* $\langle id, g, M\rangle \in \mathbf{NSC}$*, and for all* $u \in M$*, there exists a static named graph or container having the name* u *in* Σ*.*

To generate an LDP dataset from a static LDP dataset, a bijective mapping $m : \mathbf{ID} \to \mathbf{IRI}$ is required to replace the temporary identifiers to the final IRIs of LDP resources.

Dynamic LDP Dataset

Figure 4.12 shows an overview of how we deal we this change by generating *dynamic LDP datasets* from a design document that store LDP resources and instructions to generate their graphs rather than their graphs themselves. Then, at deployment, the dynamic LDP dataset may either be used to generate the final LDP dataset or only the graph of the requested LDP resource by dynamically interpreting it at query time. In this way, if a change in resources' description occurs in a data source, the LDP resources affected would contain a valid RDF graph.

Dynamic LDP datasets extend on static LDP datasets and rather than materializing RDF graphs of containers and non-containers, instructions to generate these graphs are stored in the form of CONSTRUCT queries. The formal definition of a dynamic LDP dataset is given below.

Definition 18 (Dynamic LDP dataset)**.** *A* dynamic LDP dataset *is a pair* $\langle \mathbf{dNC}, \mathbf{dC}\rangle$ *where* $\mathbf{dNC}$ *is a set of* dynamic non-container structures *and* $\mathbf{dC}$ *is a set of* dynamic container structures*. A dynamic non-container structure* $dnc \in dNC$ *is interpreted as non-container and is a pair* $\langle id, gdes\rangle$ *such that* $id \in \mathbf{ID}$ *(called the non-container name) and* $gdes$ *is a* `GraphDescription` *(described below). A dynamic container*

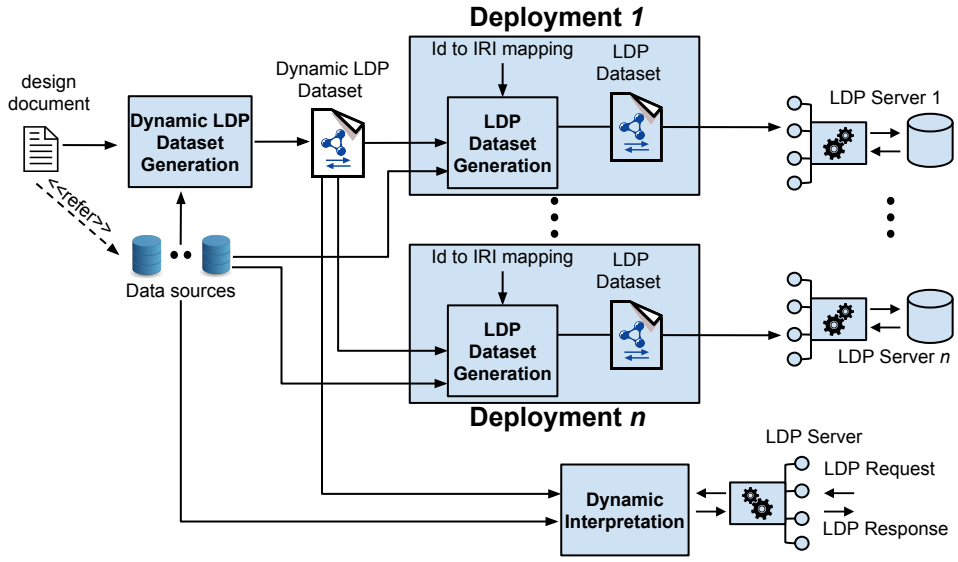

FIGURE 4.12: Application of dynamic LDP datasets

structure is a triple $(id, gdes, M)$ *and is interpreted as a container such that* $id \in \mathbf{ID}$ *(called the container name) and* $gdes$ *is a* `GraphDescription` *and* $M \in 2^{\mathbf{ID}}$*. A* `GraphDescription` *is a pair* (cq, DS) *such that* cq *is a SPARQL CONSTRUCT query and* DS *is a set of* `DataSources`*.*

In a dynamic LDP dataset ϕ*:*

- *no id appears more than once as a (dynamic non-container or container) name;*
- *for all* $\langle id, gdes, M\rangle \in \mathbf{dC}$*, and for all* $u \in M$*, there exists a dynamic container or non-container having the name* u *in* ϕ*.*

To generate an LDP dataset from a dynamic LDP dataset, in addition to a bijective mapping $m : \mathbf{ID} \rightarrow \mathbf{IRI}$ for replacing temporary identifiers of LDP resources to their final IRIs, the CONSTRUCT queries in the `GraphDescription` of LDP resources need to be evaluated to obtain their RDF graphs.

4.4 Operational Semantics

In the previous section, we presented the formal semantics of LDP-DL in a model-theoretic manner and described the LDP dataset model to store LDP resources. In this section, in Section 4.4.1, we give the operational semantics of LDP-DL by providing a family of the evaluation algorithms for LDP-DL constructs that implements the formal semantics to generate an LDP dataset from a design document. Then, in Section 4.4.2, we describe the correctness of these algorithms using the satisfaction of LDP-DL constructs presented in Section 4.2.3.

4.4.1 Evaluation Algorithms

In this section, we describe the evaluation of a design document to generate an LDP dataset using a family of algorithms for LDP-DL constructs in a bottom-up fashion

starting with DataSources and continuing with ResourceMaps, ContainerMaps and finally NonContainerMaps. We assume the existence of a global state Σ, representing the LDP dataset, that the algorithm have access to and which they construct.

Evaluation of DataSource

The evaluation of a DataSource ds is given by the function $[\![.]\!]_{source}$ s.t.

$$[\![ds]\!]_{source} = \begin{cases} \mathsf{deref}(u_{\mathsf{loc}}), & \text{if ds=}(u_{\mathsf{ds}},u_{\mathsf{loc}}) \\ \mathsf{deref}(u_{\mathsf{lr}})(\mathsf{deref}(u_{\mathsf{loc}})), & \text{if ds=}(u_{\mathsf{ds}},u_{\mathsf{loc}},u_{\mathsf{lr}}) \end{cases}$$

As mentioned before, we restrict the definition of a DataSource only to two forms and therefore, we define the evaluation only for these forms. Abusive notation, we define the evaluation of a set of DataSources $\mathbf{DS}$ as

$$[\![\mathbf{DS}]\!]_{source} = \mathrm{Merge}(\{[\![ds]\!]_{source} \mid ds \in \mathbf{DS}\})$$

Evaluation of ResourceMap

$Eval_{rm}$ (Algorithm 1) performs the evaluation of a ResourceMap $rm = \langle u_{\mathsf{rm}}, \mathsf{qp}, \mathsf{cq}, \mathbf{DS}\rangle$ with respect to an ancestor list $\vec{p}$. When evaluating the ResourceMaps of ContainerMaps found at the top of the design document, both $\vec{p}$ and Σ is empty.

Algorithm 1 Evaluation of a ResourceMap rm

1: **procedure** $Eval_{rm}(rm, \vec{p})$
2: $result \leftarrow \varnothing$
3: $g_s \leftarrow [\![\mathbf{DS}]\!]_{source}$ //perform merge of all RDF graph obtain from every data source
4: **for all** $i \in (1..len(\vec{p}))$ **do** //bind ancestor variables
5: $\mu_{\vec{p}}(\pi_i) = \vec{p}[i]$
6: **end for**
7: $\Omega \leftarrow \Pi_\rho([\![\mathsf{qp}]\!]_{g_s} \bowtie \{\mu_{\vec{p}}\})$ //get related resources
8: $result \leftarrow \varnothing$ //structure to hold new resources generated from rm
9: **for all** $\mu \in \Omega$ **do** //Iterate over all mappings
10: **if** $\mu(\rho) \neq \epsilon$ **then** //when there exist a related resource
11: $\mu_\rho(\rho) = \mu(\rho)$
12: **end if**
13: $n \leftarrow geniri(\mu(\rho), \dots)$ //generate IRI of new resource
14: $g \leftarrow \mathsf{gt}_{\mathsf{cq}}([\![\mathsf{qp}_{\mathsf{cq}}]\!]_{g_s} \bowtie \{\nu \leftarrow n\} \bowtie \{\mu_\rho\} \bowtie \{\mu_{\vec{p}}\})$ /*generate graph of new resource using $\mathsf{cq} = \langle \mathsf{gt}_{\mathsf{cq}}, \mathsf{qp}_{\mathsf{cq}}\rangle$ from rm*/
15: $result \leftarrow result \cup \{(n, \mu(\rho), g)\}$
16: **end for**
17: return $result$
18: **end procedure**

Initially, the DataSources of rm are evaluated as described in the previous section(Line 3). Then, the set of related resources is extracted (Line 7) by evaluating the query pattern of rm over the RDF source g_s. Since the query pattern may use ancestor variables, binding for the latter variables are created (Line 4 - 6) and supplied to the evaluation of the query pattern through $\mu_{\vec{p}}$ (Line 7). On obtaining the related resources in Ω, for each of them, a new LDP RDF source is created together with its IRI and graph. The IRI is generated using the function $geniri$ (Line 13). We do not

explicitly define this function and its parameters are only for illustration. What is important is that it respect the `unicity constraint` defined in Section 4.2.3.

To generate the graph of the new resource, the CONSTRUCT query cq of rm is evaluated on g_s. With respect to a newly created LDP resource, cq may contain references to its ancestors, to the resource itself or its related resource. This is why, evaluating cq, ν, ρ and $\mu_{\vec{p}}$ are supplied as part of the evaluation process of cq (Line 14). Finally for all new resources generated, their IRI, related resource and graph are returned.

Evaluation of `NonContainerMap`

$Eval_{nm}$ (Algorithm 2) evaluates a `NonContainerMap` $nm = \langle u_{\text{nm}}, \mathbf{RM} \rangle$ with respect to an ancestor list $\vec{p}$.

When evaluating a `NonContainerMap` nm, initially, all its `ResourceMaps` are evaluated using $Eval_{rm}$, defined in the previous section, to generated a set of triples (n, r, g) where n is the IRI of new LDP resource, r its related resource and g its RDF graph (Line 3 - 7).

Using this set, all new LDP resources are typed as non-containers. They are finally added to the LDP dataset (Line 3). As we mentioned before, here, the LDP dataset (Σ) can be accessed directly as it is a global state.

Algorithm 2 Evaluation of Non-ContainerMap NM

```
1:  procedure Eval_nm(nm, p⃗)
2:      result ← ∅
3:      for all rm ∈ RM do
4:          for all (n, r, g) ∈ Eval_rm(rm, p⃗) do //evaluate nm's ResourceMaps
5:              result ← result ∪ {(n, g)} //generate the non-container
6:          end for
7:      end for
8:      Σ ← Σ ∪ result  //add non-containers generated from nm to the global LDP dataset
9:      return result
10: end procedure
```

Evaluation of a `ContainerMap`

$Eval_{cm}$ (Algorithm 3) evaluates a `ContainerMap` $cm = \langle u_{\text{cm}}, \mathbf{RM}, \mathbf{CM}, \mathbf{NM} \rangle$ with respect to ancestor list $\vec{p}$.

When evaluating a `ContainerMap` cm, initially, all its `ResourceMaps` are evaluated to generated a set of triples (n, r, g) where n is the IRI of new LDP resource, r its related resource and g its graph (Line 3 - 7). For every (n, r, g), a container will be eventually generated. Therefore, we generate the members for every of these containers by evaluating the `NonContainerMaps` (Line 6 - 10) and `ContainerMaps` (Line 11 - 15) of cm in their context with their related resources appended (i.e. $\vec{p} :: r$) to the ancestor list. After all the containers and their members have been generated, they are added to the LDP dataset (Line 19).

Algorithm 3 Evaluation of ContainerMap CM

```
1:  procedure Eval_cm(cm, p⃗)
2:      result ← ∅
3:      for all rm ∈ RM do
4:          for all (n, r, g) ∈ Eval_rm(rm, p⃗) do /*Iterate over all results from evaluation
    of all ResourceMaps */
5:              M ← ∅ //Initialize set of member for current container n
6:              for all nm ∈ NM do
7:                  for all (n′, g′) ∈ Eval_nm(nm, p⃗ :: r) do //Evaluate nm in the context of n
8:                      M ← M ∪ {n′} /*add non-containers generated as members*/
9:                  end for
10:             end for
11:             for all cm′ ∈ CM do //Evaluate cm′ in the context of n
12:                 for all (n′, g′, M′) ∈ Eval_cm(cm′, p⃗ :: r) do
13:                     M ← M ∪ {n′} /*add containers generated as members*/
14:                 end for
15:             end for
16:             result = result ∪ {(n, g, M)}
17:         end for
18:     end for
19:     Σ ← Σ ∪ result//add containers generated from nm to the global LDP dataset
20:     return result
21: end procedure
```

Evaluation of a Design Document

The evaluation of a design document $\delta = (\mathbf{CM}, \mathbf{NM})$ consists of evaluating its `ContainerMap` in $\mathbf{CM}$ and `NonContainerMaps` in $\mathbf{NM}$. At the beginning, the LDP dataset is initialized to $\varnothing$ and an empty ancestor list is created. The LDP dataset Σ is the global state that is modified when the `ContainerMaps` and `NonContainerMaps` of the design document are evaluated. The evaluation of the design document starts by evaluating its top-level `ContainerMaps` and `NonContainerMaps` with respect to an empty ancestor list (Line 3 - 8).

Algorithm 4 Evaluation of design document

```
1: procedure Eval(δ)
2:     Global Σ ← ∅ //Initialize LDP Dataset
3:     for all cm ∈ CM do//Evaluate ContainerMaps
4:         Eval_cm(cm, ∅)
5:     end for
6:     for all nm ∈ NM do//Evaluate NonContainerMaps
7:         Eval_nm(nm, ∅)
8:     end for
9: end procedure
```

4.4.2 Proof of Correctness

In the previous section, we provided an algorithm for evaluating a design document. In this section, we demonstrate the correctness of this algorithm. Based on the definition of correctness (Definition 16), an algorithm is correct if it generates an

LDP dataset that is valid from a design document δ. For this LDP dataset to be valid, there must exist an interpretation $\mathcal{I}$ that satisfy δ (Definition 15). Therefore, to demonstrate the correctness of our algorithms, we do the following:

- define an interpretation $\mathcal{I}^\delta$
- show that $\mathcal{I}^\delta$ satisfies δ using conditions from the satisfaction of LDP-DL constructs given in Section 4.2.3
- show that the evaluation of δ using $\mathcal{I}^\delta$ indeed produces a valid LDP dataset.

Definition of $\mathcal{I}^\delta$

Using the definition of an LDP-DL interpretation given in Section 4.2.3, we instantiate $\mathcal{I}^\delta = \langle \Delta^\mathcal{I}_\delta, \mathcal{C}_\delta, \mathcal{N}_\delta, \mathcal{R}_\delta, \mathcal{S}_\delta, \cdot^{\mathcal{I}^\delta}, \mathcal{I}^\delta_\mathcal{C}, \mathcal{I}^\delta_\mathcal{N}, \mathcal{I}^\delta_\mathcal{R}, \mathcal{I}^\delta_\mathcal{S} \rangle$ such that:

- $\Delta^\mathcal{I}_\delta = \mathbf{IRI}$
- $\mathcal{C}_\delta = \{u_{\mathsf{cm}} \in \mathbf{IRI} | \exists cm = (u_{\mathsf{cm}}, \mathbf{RM}, \mathbf{CM}, \mathbf{CM}), cm \in \delta\}$
- $\mathcal{N}_\delta = \{u_{\mathsf{nm}} \in \mathbf{IRI} | \exists nm = (u_{\mathsf{nm}}, \mathbf{RM}), nm \in \delta\}$
- $\mathcal{R}_\delta = \{u_{\mathsf{rm}} \in \mathbf{IRI} | \exists rm = (u_{\mathsf{rm}}, \mathsf{qp}, \mathsf{cq}, \mathbf{DS}), rm \in \delta\}$
- $\mathcal{S}_\delta = \{u_{\mathsf{ds}} \in \mathbf{IRI} | \exists ds = (u_{\mathsf{ds}}, u_{\mathsf{loc}}), ds \in \delta \vee \exists ds = (u_{\mathsf{ds}}, u_{\mathsf{loc}}, u_{\mathsf{lr}}), ds \in \delta\}$
- $\cdot^{\mathcal{I}\delta}$ is the identity function;
- $\forall u_{\mathsf{cm}} \in \mathcal{C}_\delta, \exists cm = (u_{\mathsf{cm}}, \mathbf{RM}, \mathbf{CM}, \mathbf{NM}) \in \delta$,
  - $\mathcal{I}^\delta_\mathcal{C}((u_{\mathsf{cm}})^{\mathcal{I}\delta}, \vec{p}) = Eval_{cm}(cm, \vec{p})$
- $\forall u_{\mathsf{nm}} \in \mathcal{N}_\delta, \exists nm = (u_{\mathsf{nm}}, \mathbf{RM}) \in \delta$
  - $\mathcal{I}^\delta_\mathcal{N}((u_{\mathsf{nm}})^{\mathcal{I}\delta}, \vec{p}) = Eval_{nm}(nm, \vec{p})$
- $\forall u_{\mathsf{rm}} \in \mathcal{R}_\delta, \exists rm = (u_{\mathsf{rm}}, \mathsf{cq}, \mathsf{qp}, \mathbf{DS}) \in \delta$
  - $\mathcal{I}^\delta_\mathcal{R}((u_{\mathsf{rm}})^{\mathcal{I}\delta}, \vec{p}) = Eval_{rm}(rm, \vec{p})$
- $\forall u_{\mathsf{ds}} \in \mathcal{S}_\delta, \exists ds = (u_{\mathsf{ds}}, u_{\mathsf{loc}}, u_{\mathsf{lr}}) \in \delta$
  - $\mathcal{I}^\delta_\mathcal{S}((u_{\mathsf{ds}})^{\mathcal{I}\delta}) = [\![ds]\!]_{source}$

Now that we have defined the different parts of $\mathcal{I}^\delta$, let us show that $\mathcal{I}^\delta \models \delta$ in the next section.

Satisfaction of Design Document δ Using $\mathcal{I}^\delta$

We show that $\mathcal{I}^\delta$ satisfies δ by demonstrating that it satisfies the conditions with respect to `ContainerMaps`, `NonContainerMaps`, `ResourceMaps` and `DataSources` given in Section 4.2.3. We do so in a bottom-up approach starting from `DataSources`.

Satisfaction of `DataSources` by $\mathcal{I}^\delta$: Per Definition 7, the satisfaction of a `DataSource` ds by $\mathcal{I}^\delta$ depends on its two possible form. First, if $ds = \langle u_{\mathsf{ds}}, u_{\mathsf{loc}} \rangle$ then $\mathcal{I}^\delta$ satisfies ds iff $\mathcal{I}^\delta_{\mathcal{S}}(u^{\mathcal{I}^\delta}_{\mathsf{ds}}) = \mathsf{deref}(u_{\mathsf{loc}})$. Otherwise, if $ds = \langle u_{\mathsf{ds}}, u_{\mathsf{loc}}, u_{\mathsf{lr}} \rangle$ then $\mathcal{I}^\delta$ satisfies ds iff $\mathcal{I}^\delta_{\mathcal{S}}(u^{\mathcal{I}^\delta}_{\mathsf{ds}}) = \mathsf{deref}(u_{\mathsf{lr}})(\mathsf{deref}(u_{\mathsf{loc}}))$. $\mathcal{I}^\delta_{\mathcal{S}}$ is given by $[\![ds]\!]_{source}$ that implements the latter two conditions depending on the form of ds as shown in Section 4.4.1, therefore $\mathcal{I}^\delta \models \mathbf{ds}$.

Satisfaction of `ResourceMaps` by $\mathcal{I}^\delta$: In $\mathcal{I}^\delta$, `ResourceMaps` are interpreted by $\mathcal{I}^\delta_{\mathcal{R}}$. $\mathcal{I}^\delta$ satisfies a `ResourceMap` $rm = (u_{\mathsf{rm}}, \mathsf{qp}, \mathsf{cq}, \mathbf{DS})$ with respect to an ancestor list $\vec{p}$ if the following three condition.

Condition 1: We want to show that $\mathcal{I}^\delta \models \mathbf{DS}$. As shown above, $\mathcal{I}^\delta$ satisfies this condition.

Condition 2: We want to show that We want to show that if $r \in \Pi_\rho([\![\mathsf{qp}]\!]_{g_s} \bowtie \{\mu_{\vec{p}}\})$ then there exists $\langle n, r, g \rangle \in \mathcal{I}^\delta_{\mathcal{R}}(u^{\mathcal{I}^\delta}_{\mathsf{rm}}, \vec{p})$.

Let rm_1 =(u_{rm1},qp_1,cq_1,$\mathbf{DS_1}$) be a `ResourceMap` and $\vec{p_1}$ be an ancestor list.By definition of $\mathcal{I}^\delta$ in Section 4.4.2, $\mathcal{I}^\delta_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^\delta}, \vec{p}) = Eval_{rm}(rm, \vec{p})$. When evaluating rm_1 in $Eval_{rm}$, g_{s1} is produced using $\mathbf{DS_1}$ on Line 3 (Algorithm 1) and $\mu_{\vec{p_1}}$ containing ancestor variable mappings is created using $\vec{p_1}$ on Line 4 (Algorithm 1). On Line 7 (Algorithm 1), $\Pi_\rho([\![\mathsf{qp}_1]\!]_{g_{s1}} \bowtie \{\mu_{\vec{p_1}}\})$ is evaluated to generate a list of related resource. Let r_1 be one such related resource. Then, for r_1, an n_1 and g_1 is generated in the loop (Algorithm 1, Line 9 - 16). Finally, (n_1, r_1, g_1) is generated from $Eval_{rm}(rm_1, \vec{p_1})$ that implies that $(n_1, r_1, g_1) \in \mathcal{I}^\delta_{\mathcal{R}}(\mathsf{iri}(rm_1)^{\mathcal{I}^\delta}, \vec{p_1})$.

Therefore, this condition is satisfied as indeed for every $r \in \Pi_\rho([\![\mathsf{qp}]\!]_{g_s} \bowtie \{\mu_{\vec{p}}\})$, there is a $\langle n, r, g \rangle \in \mathcal{I}^\delta_{\mathcal{R}}(u^{\mathcal{I}^\delta}_{\mathsf{rm}}, \vec{p})$

Condition 3: We want to show that if $\langle n, r, g \rangle \in \mathcal{I}^\delta_{\mathcal{R}}(u^{\mathcal{I}^\delta}_{\mathsf{rm}}, \vec{p})$ then $r \in \Pi_\rho([\![\mathsf{qp}]\!]_{g_s} \bowtie \{\mu_{\vec{p}}\})$ and $g \supseteq \mathsf{gt}_{\mathsf{cq}}([\![\mathsf{qp}_{\mathsf{cq}}]\!]_{g_s} \bowtie \{\mu_\nu\} \bowtie \{\mu_\rho\} \bowtie \{\mu_{\vec{p}}\})$.

Let rm_1 =(u_{rm1},qp_1,cq_1,$\mathbf{DS_1}$) is a `ResourceMap` and $\vec{p_1}$ be an ancestor list. g_{s1} is produced using $\mathbf{DS_1}$ on Line 3 (Algorithm 1) and $\mu_{\vec{p_1}}$ containing ancestor variable mappings is created using $\vec{p_1}$ on Line 4 (Algorithm 1). By definition of $\mathcal{I}^\delta$ in Section 4.4.2, $\mathcal{I}^\delta_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^\delta}, \vec{p}) = Eval_{rm}(rm, \vec{p})$. Let (n_1, r_1, g_1) be a triple generated by $Eval_{rm}$ (rm_1,$\vec{p_1}$). (n_1, r_1, g_1) is generated in the loop (Algorithm 1, Line 15). When it is being generated in that loop, $\mu = r_1$ (Algorithm 1, Line 9) is always an element of Ω that is indeed obtained only from the expression $\Pi_\rho([\![\mathsf{qp}_1]\!]_{g_{s1}} \bowtie \{\mu_{\vec{p_1}}\})$ (Algorithm 1, Line 7). Also, the graph g is directly generated on Line 14 (Algorithm 1) using the expression $\mathsf{gt}_{\mathsf{cq}_1}([\![\mathsf{qp}_{\mathsf{cq}_1}]\!]_{g_{s1}} \bowtie \{\nu \leftarrow n\} \bowtie \{\mu_\rho\} \bowtie \{\mu_{\vec{p_1}}\})$. No other triples is added to g. Thus, if $(n_1, r_1, g_1) \in \mathcal{I}^\delta_{\mathcal{R}}(rm_1^{\mathcal{I}^\delta}, \vec{p_1})$, both r_1 and g_1 satisfies the constraints in the condition.

Therefore, this condition is satisfied as indeed for every $\langle n, r, g \rangle \in \mathcal{I}^\delta_{\mathcal{R}}(u^{\mathcal{I}^\delta}_{\mathsf{rm}}, \vec{p})$, there is a $r \in \Pi_\rho([\![\mathsf{qp}]\!]_{g_s} \bowtie \{\mu_{\vec{p}}\})$ with $g \supseteq \mathsf{gt}_{\mathsf{cq}}([\![\mathsf{qp}_{\mathsf{cq}}]\!]_{g_s} \bowtie \{\mu_\nu\} \bowtie \{\mu_\rho\} \bowtie \{\mu_{\vec{p}}\})$

Since the above conditions are satisfied, $\mathcal{I}^\delta$ satisfies a `ResourceMap` with respect to a ancestor list $\vec{p}$ (i.e. $\mathcal{I}^\delta, \vec{p} \models rm$).

Satisfaction of NonContainerMaps by $\mathcal{I}^{\delta}_{\mathcal{N}}$: In $\mathcal{I}^{\delta}$, NonContainerMaps are interpreted by $\mathcal{I}^{\delta}_{\mathcal{N}}$. $\mathcal{I}^{\delta}$ satisfies a NonContainerMap $nm = (u_{\mathsf{nm}}, \mathbf{RM})$ with respect to an ancestor list $\vec{p}$ if the following three condition are satisfied. When demonstrating the satisfactions of the conditions below, we use the definition of $\mathcal{I}^{\delta}$ in Section 4.4.2 where $\mathcal{I}^{\delta}_{\mathcal{N}}(\mathsf{iri}(nm)^{\mathcal{I}^{\delta}}, \vec{p}) = Eval_{nm}(nm, \vec{p})$ and for any ResourceMap rm, $\mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^{\delta}}, \vec{p}) = Eval_{rm}(rm, \vec{p})$.

Condition 1: We want to show that $\mathcal{I}, \vec{p} \models \mathbf{RM}$. As shown above, $\mathcal{I}^{\delta}$ satisfies this condition.

Condition 2: We want to show that if $\langle n, g\rangle \in \mathcal{I}^{\delta}_{\mathcal{N}}(u^{\mathcal{I}^{\delta}}_{\mathsf{nm}}, \vec{p})$ then there exists $rm \in \mathbf{RM}, \langle n, r, g'\rangle \in \mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^{\delta}}, \vec{p})$ such that $g' \subseteq g$.

Let nm_1 =(u_{nm1},rm_1) be a NonContainerMap and $\vec{p_1}$ an ancestor list. $Eval_{nm}$ (nm_1,$\vec{p_1}$) returns a set of non-containers $\langle n, g\rangle$. Let $\langle n_1, g_1\rangle$ be such a non-container. For $\langle n_1, g_1\rangle$ to be generated on Line 5 (Algorithm 2), there must necessarily be a $\langle n_1, r_1, g_1\rangle$ generated in $Eval_{rm}$ (rm_1,$\vec{p_1}$) on Line 4 (Algorithm 2) where r_1 is the related resource that LDP resource n_1 describes. Also, the graph in $\langle n_1, g_1\rangle$ is the same as the one in $\langle n_1, r_1, g_1\rangle$ as we use the latter triple directly without adding any triples to g_1. Thus, if $\langle n_1, g_1\rangle \in \mathcal{I}^{\delta}_{\mathcal{N}}(\mathsf{iri}(nm_1)^{\mathcal{I}^{\delta}}, \vec{p_1})$, then $\langle n_1, r_1, g_1\rangle \in \mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm_1)^{\mathcal{I}^{\delta}}, \vec{p_1})$

Therefore, this condition is satisfied as indeed for every $\langle n, g\rangle$ in $\mathcal{I}^{\delta}_{\mathcal{N}}(u^{\mathcal{I}^{\delta}}_{\mathsf{nm}}, \vec{p})$, there is a $\langle n, r, g\rangle$ in $\mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^{\delta}}, \vec{p})$ where $rm \in \mathbf{RM}$

Condition 3: We want to show that if there exists $rm \in \mathbf{RM}, \langle n, r, g'\rangle \in \mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^{\delta}}, \vec{p})$, then there exists a unique $\langle n, g\rangle \in \mathcal{I}^{\delta}_{\mathcal{N}}(u^{\mathcal{I}^{\delta}}_{\mathsf{nm}})$ such that $g' \subseteq g$.

Let nm_1 =(u_{nm1},rm_1) be a NonContainerMap and $\vec{p_1}$ an ancestor list. When evaluating ResourceMaps in $Eval_{nm}$ on Line 3 (Algorithm 2), for a particular (n_1, r_1, g_1) in $Eval_{rm}$ (nm_1,$\vec{p_1}$), only one (n_1, g_1) is created on Line 5 (Algorithm 2). Also, the same graph g_1 is used without adding any further triples to it. Thus, if $\langle n_1, r_1, g_1\rangle \in \mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm_1)^{\mathcal{I}^{\delta}}, \vec{p_1})$, then indeed there exists a unique $\langle n_1, g_1\rangle \in \mathcal{I}^{\delta}_{\mathcal{N}}(\mathsf{iri}(nm_1)^{\mathcal{I}^{\delta}})$.

Therefore, this condition is satisfied as indeed for every $\langle n, r, g\rangle$ in $\mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^{\delta}}, \vec{p})$ where $rm \in \mathbf{RM}$, there is a $\langle n, g\rangle$ in $\mathcal{I}^{\delta}_{\mathcal{N}}(u^{\mathcal{I}^{\delta}}_{\mathsf{nm}}, \vec{p})$.

Since the above conditions are satisfied, $\mathcal{I}^{\delta}$ satisfies a NonContainerMap nm with respect to an ancestor list $\vec{p}$ (i.e. $\mathcal{I}^{\delta}, \vec{p} \models nm$).

Satisfaction of ContainerMaps by $\mathcal{I}^{\delta}_{\mathcal{C}}$: In $\mathcal{I}^{\delta}$, ContainerMaps are interpreted by $\mathcal{I}^{\delta}_{\mathcal{C}}$. $\mathcal{I}^{\delta}$ satisfies a ContainerMap $cm = (u_{\mathsf{cm}}, \mathbf{RM}, \mathbf{CM}, \mathbf{NM})$ with respect to an ancestor list $\vec{p}$ if the following three condition are satisfied. When demonstrating the satisfaction of the conditions below, we use the definition of $\mathcal{I}^{\delta}$ in Section 4.4.2 where $\mathcal{I}^{\delta}_{\mathcal{C}}(\mathsf{iri}(cm)^{\mathcal{I}^{\delta}}, \vec{p}) = Eval_{cm}(cm, \vec{p})$ and for any NonContainerMap nm, $\mathcal{I}^{\delta}_{\mathcal{N}}(\mathsf{iri}(nm)^{\mathcal{I}^{\delta}}, \vec{p}) = Eval_{nm}(nm, \vec{p})$ and for any ResourceMap rm, $\mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^{\delta}}, \vec{p}) = Eval_{rm}(rm, \vec{p})$ and for any

Condition 1: We want to show that $\mathcal{I}^{\delta}, \vec{p} \models \mathbf{RM}$. As shown above, $\mathcal{I}^{\delta}$ satisfies this condition.

Condition 2: We want to show that if $\langle n, g, M\rangle \in \mathcal{I}^{\delta}_{\mathcal{C}}(u^{\mathcal{I}}_{\mathsf{cm}})$ then there exists $rm \in \mathbf{RM}, \langle n, r, g'\rangle \in \mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^{\delta}}, \vec{p})$ such that $g' \subseteq g$.

Let cm_1 =(u_{cm1},rm_1,$\varnothing$,$\varnothing$) be a `ContainerMap` and $\vec{p_1}$ be an ancestor list. $Eval_{cm}$ (cm_1,$\vec{p_1}$) returns a set of containers $\langle n, g, M\rangle$. Suppose $\langle n_1, g_1, M_1\rangle$ is one such container. For container $\langle n_1, g_1, M_1\rangle$ to be generated on Line 16 (Algorithm 3), there must necessarily be a $\langle n_1, r_1, g_1\rangle$ generated in $Eval_{rm}$ (rm_1,$\vec{p_1}$) on Line 4 (Algorithm 3) where r_1 is the related resource that container n describes. Also, the graph g_1 of container n_1 is the same as the one in $\langle n_1, r_1, g_1\rangle$ as we use the latter triple directly without adding any triples in g_1. The set of members M_1 of container n_1 will be empty since it does not have any child `ContainerMaps` or `NonContainerMaps`. Thus, if $\langle n_1, g_1, M_1\rangle$ is generated from $\mathcal{I}^{\delta}_{\mathcal{C}}(u^{\mathcal{I}}_{\mathsf{cm}})$, then there must be an $\langle n_1, r_1, g_1\rangle \in \mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^{\delta}}, \vec{p})$

Therefore, this condition is satisfied as indeed for every $\langle n, g, M\rangle$ in $\mathcal{I}^{\delta}_{\mathcal{C}}(u^{\mathcal{I}^{\delta}}_{\mathsf{cm}}, \vec{p})$, there is a $\langle n, r, g\rangle$ in $\mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^{\delta}}, \vec{p})$ where $rm \in \mathbf{RM}$

Condition 3: We want to show that if there exists $rm \in \mathbf{RM}, \langle n, r, g'\rangle \in \mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^{\delta}}, \vec{p})$, then there exists a unique $\langle n, g, M\rangle \in \mathcal{I}_{\mathcal{C}}(u^{\mathcal{I}}_{\mathsf{cm}})$ such that $g' \subseteq g$.

Let cm_1 =(u_{cm1},rm_1,$\varnothing$,$\varnothing$) be a `ContainerMap` and $\vec{p_1}$ an ancestor list. When evaluating `ResourceMaps` in $Eval_{cm}$ on Line 3 (Algorithm 3), for a particular (n_1, r_1, g_1) in $Eval_{rm}$ (rm_1,$\vec{p_1}$), only one (n_1, g_1, M_1) is created on Line 16 (Algorithm 3). Also, the same graph g_1 is used without adding any further triples to it.

Therefore, this condition is satisfied as indeed for every $\langle n, r, g\rangle$ in $\mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^{\delta}}, \vec{p})$ where $rm \in \mathbf{RM}$, there is a $\langle n, g, M\rangle$ in $\mathcal{I}^{\delta}_{\mathcal{C}}(u^{\mathcal{I}^{\delta}}_{\mathsf{cm}}, \vec{p})$.

Condition 4: We want to show that for all $rm \in \mathbf{RM}$ and for all $\langle n, r, g\rangle \in \mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^{\delta}})$ and for all $\langle n, g, M\rangle \in \mathcal{I}^{\delta}_{\mathcal{C}}(u^{\mathcal{I}^{\delta}}_{\mathsf{cm}}, \vec{p})$, we have $\mathcal{I}^{\delta}, \vec{p} :: r \models \mathbf{NM}$ and $M \supseteq \{n \mid \exists nm \in \mathbf{NM} \wedge \exists \langle n, g\rangle \in \mathcal{I}^{\delta}_{\mathcal{N}}(\mathsf{iri}(nm)^{\mathcal{I}^{\delta}}, \vec{p} :: r)\}$.

Let cm_1 =(u_{cm1},{rm_1},{nm_1},{}) be a `ContainerMap` and $\vec{p_1}$ an ancestor list. Suppose that (n_1, r_1, g_1) is obtained from the evaluation of rm_1 on Line 3 (Algorithm 3). For this triple, only one container (n_1, g_1, M_1) is generated using it on Line 16 (Algorithm 3). With respect to the latter container, the `NonContainerMap` nm_1 is evaluated on Line 7 (Algorithm 3) in its context by appending the ancestor list $\vec{p_1}$ with its related resource r_1. The IRI of non-containers in $Eval_{nm}$ (nm_1,$\vec{p_1} :: r_1$) are then added to M_1 on Line 8 (Algorithm 3).

Therefore, this condition is satisfied as indeed for every container (n, g, M) in $\mathcal{I}^{\delta}_{\mathcal{C}}(cm, \vec{p})$, M will the IRIs of all non-containers generated from the `NonContainerMaps` of cm.

Condition 5: We want to show that for all $rm \in \mathbf{RM}$ and for all $\langle n, r, g\rangle \in \mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^{\delta}})$, and for all $\langle n, g, M\rangle \in \mathcal{I}^{\delta}_{\mathcal{C}}(u^{\mathcal{I}^{\delta}}_{\mathsf{cm}}, \vec{p})$, we have $\mathcal{I}^{\delta}, \vec{p} :: r \models \mathbf{CM}$ and $M \supseteq \{n \mid \exists cm \in \mathbf{CM} \wedge \exists \langle n, g, M'\rangle \in \mathcal{I}^{\delta}_{\mathcal{C}}(\mathsf{iri}(rm)^{\mathcal{I}^{\delta}}, \vec{p} :: r)\}$.

Let cm_1 =(u_{cm1},{rm_1},$\varnothing$,{cm_{11}}) be a `ContainerMap` and $\vec{p_1}$ an ancestor list. Suppose that (n_1, r_1, g_1) is obtained from the evaluation of rm_1 on Line 3 (Algorithm 3). For this triple, only one container (n_1, g_1, M_1) is generated using it on Line 16 (Algorithm 3). With respect to the latter container, the `ContainerMap` cm_{11} is evaluated on Line 12 (Algorithm 3) in its context by appending the ancestor list $\vec{p_1}$ with its related resource r_1. $Eval_{cm}$ (cm_{11},$\vec{p_1} :: r_1$) may return a number of containers. The IRI of all these containers are added to M_1 on Line 13 (Algorithm 3).

Therefore, this condition is satisfied as indeed for every container (n, g, M) in $\mathcal{I}^{\delta}_{\mathcal{C}}(cm, \vec{p})$, M will contain containers generated from the ContainerMaps of cm.

Condition 6: We want to show that for all $rm \in \mathbf{RM}$ and for all $\langle n, r, g\rangle \in \mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^{\delta}})$, and for all $\langle n, g, M\rangle \in \mathcal{I}^{\delta}_{\mathcal{C}}(u^{\mathcal{I}^{\delta}}_{\mathsf{cm}}, \vec{p}), n' \in M_n \Rightarrow (\exists nm \in \mathbf{NM}, \exists g', \langle n', g'\rangle \in \mathcal{I}^{\delta}_{\mathcal{N}}(\mathsf{iri}(nm)^{\mathcal{I}^{\delta}}, \vec{p} :: r)) \vee (\exists cm' \in \mathbf{CM}, \exists g', \exists M', \langle n', g', M'\rangle \in \mathcal{I}^{\delta}_{\mathcal{C}}(\mathsf{iri}(cm')^{\mathcal{I}^{\delta}}, \vec{p} :: r)$.

Let cm_1 =(u_{cm1},{rm_1},{nm_1},{cm_{11}}) be a ContainerMap and $\vec{p_1}$ an ancestor list. Suppose that (n_1, r_1, g_1) is obtained from the evaluation of rm_1 on Line 3 (Algorithm 3) and the container (n_1, g_1, M_1) is generated using it on Line 16 (Algorithm 3). M_1 can contain only IRIs of containers and non-containers generated from cm_1 and nm_1. Because, as we can see in Algorithm 3, there are only two places where IRIs can be added to M_1. The first one is when evaluating the ContainerMaps of cm (Algorithm 3, Line 11 - 15). The second one is when evaluating the NonContainerMaps of cm (Algorithm 3, Line 6 - 10). Thus, members of containers generated from cm_1 will only contain containers and non-containers generated from cm_{11} and nm_1 with the proper ancestor list.

Therefore, this condition is satisfied as for all containers (n, g, M) in $\mathcal{I}^{\delta}_{\mathcal{C}}(u^{\mathcal{I}^{\delta}}_{\mathsf{cm}}, \vec{p})$, the elements of M can only be IRIs of containers or non-containers in the interpretation of the ContainerMaps or NonContainerMaps of cm respectively.

Since the above conditions are satisfied, $\mathcal{I}^{\delta}$ satisfies a ContainerMap m with respect to an ancestor list $\vec{p}$ (i.e. $\mathcal{I}^{\delta}, \vec{p} \models cm$).

Satisfaction of a design document by $\mathcal{I}^{\delta}$: $\mathcal{I}^{\delta}$ satisfies a design document $\delta = (\mathbf{CM}, \mathbf{NM})$ with respect to an ancestor list $\vec{p}$ if the following condition are satisfied.

Condition 1: We want to show that $\mathcal{I}^{\delta}, \emptyset \models \mathbf{CM}$.
$\mathcal{I}^{\delta}$ satisfies this condition as $\forall cm \in \mathbf{CM}, \mathcal{I}^{\delta}, \emptyset \models cm$ as shown above.

Condition 2: We want to show that $\mathcal{I}^{\delta}, \emptyset \models \mathbf{NM}$.
$\mathcal{I}^{\delta}$ satisfies this condition as $\forall nm \in \mathbf{NM}, \mathcal{I}^{\delta}, \emptyset \models nm$ as shown above.

Evaluation of design documents using $\mathcal{I}^{\delta}$

In the previous section, we have showed that $\mathcal{I}^{\delta}$ satisfy design documents. In this section, we show that the interpretation of a design document using $\mathcal{I}^{\delta}$ produces a valid LDP dataset. Given a design document $\delta = (\mathbf{CM}, \mathbf{NM})$, we want to prove that $[\![\delta]\!]_{\mathcal{I}} = Eval(\delta)$. To be able to make this proof, we make the hypothesis that the ContainerMaps in δ have no cycles and are finite. With these constraints, δ forms a tree as shown in Figure 4.13 where δ itself is the root with its immediate children be its top-level *maps that are found directly in $\mathbf{CM}$ and $\mathbf{NM}$.

Since design documents like δ form a tree, we use some concepts related to tree structures that we define below. For the remainder of this section, we use one finite design document $\delta = (\mathbf{CM}, \mathbf{NM})$ having no cycles in its ContainerMaps.

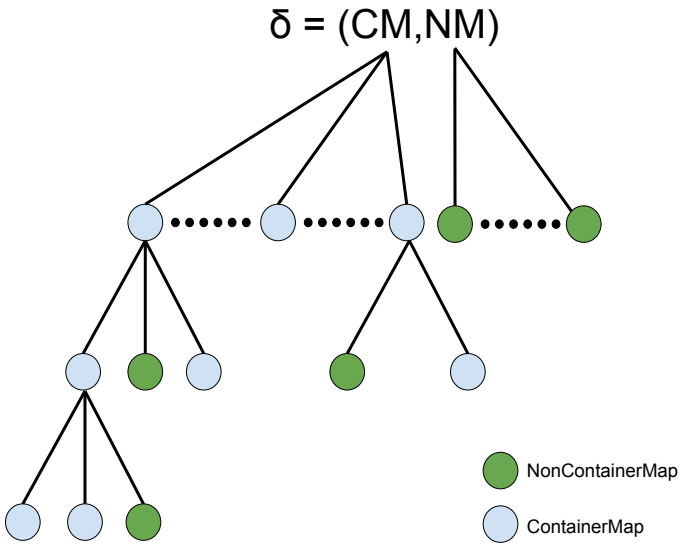

FIGURE 4.13: Abstract View of design document δ containing maps that are finite and having no cycles

In a design document, all NonContainerMaps are leaves. A ContainerMap is a leaf iff it has no child ContainerMaps.

Definition 19 (Leaf). *A* NonContainerMap *is a leaf. A* ContainerMap cm*=(*u_{cm}*,*$\mathbf{RM}$*,*$\mathbf{CM}$*,*$\mathbf{NM}$*) is a* Leaf *iff* $\mathbf{CM}$*=*$\varnothing$

In a design document, the leaves have a height of zero. The height of a ContainerMap is the length of the longest path downward to a leaf.

Definition 20 (Height). *Let* cm*=(*u_{cm}*,*$\mathbf{RM}$*,*$\mathbf{CM}$*,*$\mathbf{NM}$*) be a* ContainerMap*. If* cm *is a leaf, height(cm)=0. If* $cm' \in \mathbf{CM}$*, max(height(*cm'*))=n, then height(*cm*)=n+1*

Both ContainerMaps and NonContainerMaps are interpreted with respect to different ancestor lists that vary depending on the related resource generated by their own ResourceMaps or that of their ancestors.

Definition 21 (Ancestors List). *If* $m \in \mathbf{CM} \cup \mathbf{NM}$, ancestors$(m, \delta) = \{\varnothing\}$*. If there exist a* ContainerMap $cm' = (u_{\mathsf{cm}}, \mathbf{RM}', \mathbf{CM}', \mathbf{NM}')$ *appearing in* δ *such that* $m' \in \mathbf{CM}' \cup \mathbf{NM}'$*, then* ancestors$(m', \delta) = \{\vec{p} :: r \mid \vec{p} \in \text{ancestors}(cm', \delta), rm' \in \mathbf{RM}', \exists (n, r, g) \in Eval_{nm}(rm, \vec{p})\}$

We want to show that $[\![\delta]\!]_{\mathcal{I}} = Eval(\delta)$ by proving each inclusion separately. We start by that evaluation of the ContainerMaps are included in the generated LDP dataset.

Theorem 1. *For all* cm *in* δ*, for all* $\vec{p} \in$ ancestors(cm, δ), $[\![cm]\!]^{\vec{p}}_{\mathcal{I}^\delta} \subseteq Eval(\delta)$

Proof. Let us prove the above theorem by induction on the height of a ContainerMap $cm = (u_{\mathsf{cm}}, \mathbf{RM}, \mathbf{CM}, \mathbf{NM})$ appearing in δ.

Base Case occurs when height$(cm) = 0$

Let $e \in [\![cm]\!]^{\vec{p}}_{\mathcal{I}^\delta}$. Based on Definition 13, if $e \in [\![cm]\!]^{\vec{p}}_{\mathcal{I}^\delta}$, then either:

(1.1) $e \in \mathcal{I}^\delta_{\mathcal{C}}(\mathsf{iri}(cm)^{\mathcal{I}^\delta}, \vec{p})$, *or*

(1.2) $e \in \bigcup_{\substack{rm \in \mathbf{RM} \\ \langle n,r,g\rangle \in \mathcal{I}^\delta_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^\delta}, \vec{p}) \\ m' \in \mathbf{NM} \cup \mathbf{CM}}} [\![m']\!]^{\vec{p}::r}_{\mathcal{I}^\delta}$

Since height$(cm) = 0$, $\vec{p} = \emptyset$ and $\mathbf{CM} = \varnothing$. Let us now consider (1.1) and (1.2) individually below.

(1.1) If $e \in \mathcal{I}^\delta_{\mathcal{C}}(\mathsf{iri}(cm)^{\mathcal{I}^\delta}, \emptyset)$, then by the definition of $\mathcal{I}^\delta_{\mathcal{C}}$ in Section 4.4.2, $e \in Eval_{cm}(cm, \emptyset)$. If we consider the definition of $Eval_{cm}$ (Algorithm 3) on Line 16, we can see the only triples (n, g, M) are added to the set $result$ that is returned by $Eval_{cm}$. Therefore, $e \in Eval_{cm}(cm, \emptyset)$ implies that $e = (n, g, M)$. Also, we can notice that $result$ is always added to the LDP dataset (Σ) on Line 19. Therefore, $e = (n, g, M) \in Eval(\delta)$

(1.2) if $e \in \bigcup_{\substack{rm \in \mathbf{RM} \\ \langle n,r,g\rangle \in \mathcal{I}^\delta_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^\delta}, \vec{p}) \\ m' \in \mathbf{NM}}} [\![m']\!]^{\vec{p}::r}_{\mathcal{I}^\delta}$, then there exist a `ResourceMap` $rm' \in \mathbf{RM}$ and a `NonContainerMap` $nm' \in \mathbf{NM}$ such that $e \in \mathcal{I}^\delta_{\mathcal{N}}(\mathsf{iri}(nm')^{\mathcal{I}^\delta}, \vec{p} :: r)$ based on Definition 13. Thus, by the definition of $\mathcal{I}^\delta_{\mathcal{N}}$ in Section 4.4.2, $e \in Eval_{nm}(nm, \vec{p} :: r)$. If we consider the definition of $Eval_{nm}$ (Algorithm 2) on Line 5, we can see the only pairs (n, g) are added to the set $result$ that is returned by $Eval_{nm}$. Therefore, $e \in Eval_{nm}(nm, \vec{p} :: r)$ implies that $e = (n, g)$. Also, we can notice that $result$ is always added to the LDP dataset (Σ) on Line 8 in the definition of $Eval_{nm}$ (Algorithm 2). Therefore, $e = (n, g) \in Eval(\delta)$.

Now that we have proved that the *base case* is true, let us proceed with the definition of the inductive hypothesis.

Inductive Hypothesis Let $n \in \mathbb{N}$ such that for all `ContainerMaps` cm appearing in δ such that height$(cm) \leq n$, and for all $\vec{p} \in$ ancestors(cm, δ), $[\![cm]\!]^{\vec{p}}_{\mathcal{I}^\delta} \subseteq Eval(\delta)$. Let cm' be a `ContainerMap` appearing in δ such that $cm' = (u_{\mathsf{cm}}, \mathbf{RM}, \mathbf{CM}, \mathbf{NM})$ and height$(cm') = n + 1$ and let $\vec{p} \in$ ancestors(cm', δ).

Let $e \in [\![cm]\!]^{\vec{p}}_{\mathcal{I}^\delta}$. Based on Definition 13, if $e \in [\![cm]\!]^{\vec{p}}_{\mathcal{I}^\delta}$, then either:

(2.1) $e \in \mathcal{I}^\delta_{\mathcal{C}}(\mathsf{iri}(cm)^{\mathcal{I}^\delta}, \vec{p})$ (shown above in **(1.2)**), *or*

(2.2) $e \in \bigcup_{\substack{rm \in \mathbf{RM} \\ \langle n,r,g\rangle \in \mathcal{I}^\delta_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^\delta}, \vec{p}) \\ m' \in \mathbf{NM} \cup \mathbf{CM}}} [\![m']\!]^{\vec{p}::r}_{\mathcal{I}^\delta}$

Since we have already shown (2.1), let us proceed with showing (2.2).

(2.2) If $e \in \bigcup_{\substack{rm \in \mathbf{RM} \\ \langle n,r,g\rangle \in \mathcal{I}^{\delta}_{\mathcal{R}}(\mathsf{iri}(rm)^{\mathcal{I}^{\delta}}, \vec{p}) \\ m' \in \mathbf{NM}}} [\![m']\!]^{\vec{p}::r}_{\mathcal{I}^{\delta}}$, then there exist a $rm \in \mathbf{RM}$, $(n, r, g) \in \mathcal{I}^{\delta}_{\mathcal{R}}(rm, \vec{p})$ and $m \in \mathbf{CM} \cup \mathbf{NM}$ such that $e \in [\![m]\!]^{\vec{p}::r}$. If $m \in \mathbf{NM}$, then the case is similar to the one shown in **(1.2)**. Else, if $m \in \mathbf{NM}$, we can notice that:

- $\mathsf{height}(m) \leq n$
- $\vec{p} :: r \in \mathsf{ancestors}(m, \delta)$

Thus, from the induction hypothesis, $[\![m]\!]^{\vec{p}::r}_{\mathcal{I}^{\delta}} \subseteq Eval(\delta)$. □

We now proceed to the reverse inclusion.

Theorem 2. $Eval(\delta) \subseteq [\![\delta]\!]_{\mathcal{I}^{\delta}}$

Proof. Let us now prove the above theorem. Let $e \in Eval(\delta)$. There are 2 cases, either:

(3.1) $e = (n, g)$ is a non-container, or;

(3.2) $e = (n, g, M)$ is a container

Let us consider the case (3.1),

(3.1) If $e = (n, g)$, then it has been generated by Algorithm 2 on Line 5. So, there exist two parameters, a `NonContainerMap` nm and an ancestor list $\vec{p}$, such that $e \in Eval_{nm}(nm, \vec{p})$. Based on the definition of $\mathcal{I}^{\delta}$ in Section 4.4.2, $\mathcal{I}^{\delta}_{\mathcal{N}}(\mathsf{iri}(nm)^{\mathcal{I}^{\delta}}, \vec{p}) = Eval_{nm}(nm, \vec{p})$ and by the definition of $[\![nm]\!]^{\vec{p}}_{\mathcal{I}^{\delta}}$ (Definition 13), $\mathcal{I}^{\delta}_{\mathcal{N}}(\mathsf{iri}(nm)^{\mathcal{I}^{\delta}}, \vec{p}) = [\![nm]\!]^{\vec{p}}_{\mathcal{I}^{\delta}}$. We can prove that $[\![nm]\!]^{\vec{p}}_{\mathcal{I}^{\delta}} \subseteq [\![\delta]\!]_{\mathcal{I}^{\delta}}$. While this may seem obvious by looking at the definition of $[\![\delta]\!]_{\mathcal{I}^{\delta}}$, it depends on whether the ancestor list $\vec{p}$ used by $Eval(\delta)$ is the correct list of nm in δ. Let us specify this as a lemma and prove it.

Lemma 1. *For all $m \in \mathbf{CM} \cup \mathbf{NM}$ appearing in δ, calls to $Eval_{*m}(m, \vec{p})$ in the execution of $Eval(\delta)$ occur if and only if $\vec{p} \in \mathsf{ancestors}(m, \delta)$*

We now prove the above lemma.

Proof. Clearly it is the case at the first call to $Eval_{nm}$ and $Eval_{cm}$ on the empty ancestor list in $Eval(\delta)$(Algorithm 4). Then, the only lines where there is a recursive call to $Eval_{*m}$ are Line 7 and Line 12 in Algorithm 3.

Let the `ContainerMap` cm and the ancestor list $\vec{p}$ be parameters of a call to $Eval_{cm}$ such that $\vec{p} \in \mathsf{ancestors}(cm, \delta)$. Then, the call of $Eval_{nm}$ is made with parameters nm and $\vec{p} :: r$ (Algorithm 3, Line 7) where r is defined from a call to $Eval_{rm}$ on Line 4 (Algorithm 3) and nm is a `NonContainerMap` in cm. By definition of $\mathcal{I}^{\delta}$, $\mathcal{I}^{\delta}_{\mathcal{R}}(rm, \vec{p}) = Eval_{rm}(rm, \vec{p})$ and so $\vec{p} :: r \in \mathsf{ancestors}(nm, \delta)$. Similarly, the call to $Eval_{cm}$ is made with the right ancestor list. □

Let us now consider the case (3.2),

(3.2) if $e = (n, g, M)$, then it has been generated by Algorithm 3 with parameters cm and $\vec{p}$ such that $e \in Eval_{cm}(cm, \vec{p})$. By definition, $\mathcal{I}^{\delta}_{\mathcal{C}}$(cm,$\vec{p}$)=$Eval_{cm}$ (cm,$\vec{p}$) and by definition of $[\![cm]\!]^{\vec{p}}_{\mathcal{I}^{\delta}}$, $\mathcal{I}^{\delta}_{\mathcal{C}}(cm, \vec{p}) \subseteq [\![cm]\!]^{\vec{p}}_{\mathcal{I}^{\delta}}$. By Lemma 1, $\vec{p}$ is the ancestor list of cm, so $[\![cm]\!]^{\vec{p}}_{\mathcal{I}^{\delta}}$ appear as a term in the union in the definition of $[\![\delta]\!]_{\mathcal{I}^{\delta}}$. So $\mathcal{I}^{\delta}_{\mathcal{C}}(cm, \vec{p}) \subseteq [\![\delta]\!]_{\mathcal{I}^{\delta}}$. Therefore, $e \in [\![\delta]\!]_{\mathcal{I}^{\delta}}$

□

Theorem 3. *For all finite δ with no cycles, $Eval(\delta)$is valid with respect to δ*

Proof. For all nm in δ, for all $\vec{p} \in$ ancestors(nm, δ), $[\![nm]\!]^{\vec{p}}_{\mathcal{I}^{\delta}} = \mathcal{I}^{\delta}_{\mathcal{N}}(\text{iri}(nm)^{\mathcal{I}^{\delta}}, \vec{p})$ by definition of $\mathcal{I}^{\delta}_{\mathcal{N}}(\text{iri}(nm)^{\mathcal{I}^{\delta}}, \vec{p})$ and $\mathcal{I}^{\delta}_{\mathcal{N}}(\text{iri}(nm)^{\mathcal{I}^{\delta}}, \vec{p}) = Eval_{nm}(nm, \vec{p})$ by definition of $\mathcal{I}^{\delta}$. Moreover, $Eval_{nm}(nm, \vec{p}) \subseteq Eval(\delta)$ according to Lemma 1 and Algorithm 2. In addition of Theorem 1, this ensures that $[\![\delta]\!]_{\mathcal{I}^{\delta}} \subseteq Eval(\delta)$. Finally, Theorem 2 states that $Eval(\delta) \subseteq [\![\delta]\!]_{\mathcal{I}^{\delta}}$. So $Eval(\delta) = [\![\delta]\!]_{\mathcal{I}^{\delta}}$. □

Therefore, $Eval(\delta)$ is correct.

4.5 Summary

In this chapter, we have described the syntax and semantics of LDP-DL both in a informal and formal way. We started by first providing an overview of LDP-DL using a concrete example. Then, we have formally described the syntax of LDP-DL as well as its semantics.We have defined the semantics of our language in two different ways, model-theoretic and operational. The model-theoretic semantics have enabled us to abstract from choices left open by the LDP standard and associate an LDP dataset to an LDP-DL design. The operational semantics is an instantiation of the model-theoretic semantics to evaluate a LDP-DL design and generate an LDP dataset containing the LDP resources. Also, we have described the correctness of our evaluation algorithms using the satisfaction of LDP-DL constructs defined per the model-theoric semantics of LDP-DL. Finally, we describe the variabilities that may invalidate LDP datasets and provide two intermediary models, static and dynamic data sources, to abstract them.

Part III

Implementation & Validation

Chapter 5

Implementation

In the previous part, we have proposed the LDP generation workflow to automatize the generation of LDPs whose core is a language, LDP-DL, to define the design of LDPs. Also, we have described formally the syntax and semantics of the language. In this chapter, we describe the workflow's implementation that we refer to as the LDP generation toolkit. To this end, in Section 5.1, we provide an overview of the toolkit. We describe the concrete syntax for writing LDP-DL design documents in Section 5.2. Then, we present the tools from the LDP generation toolkit: ShapeLDP (Section 5.3) is an implementation of the LDPizer, POSTerLDP (Section 5.4) is an implementation of the LDP dataset deployer, InterLDP (Section 5.5) is an implementation of an LDP server, and, finally, HubbleLDP, an LDP browser (Section 5.6) for navigating through LDPs.

Contents

5.1 Overview of LDP Generation Toolkit

The LDP generation toolkit, whose abstract view is shown in Figure 5.1, is an implementation of the LDP generation workflow presented in Chapter 3 (Figure 3.9). ShapeLDP is an LDPizer that consumes a design document written in the concrete syntax of LDP-DL. With respect to some data sources, it generates one of the two LDP dataset variants, static or dynamic LDP datasets presented in the previous chapter, to deal with static or dynamic data sources. The LDP dataset can then be deployed on a server either by using POSTerLDP or InterLDP. POSTerLDP is an LDP dataset deployer that deploys LDP resources from the (dynamic) LDP dataset to any conforming LDP server. InterLDP is an LDP server that can directly consume a (dynamic) LDP dataset. Finally, HubbleLDP is an LDP browser that can be used to navigate through LDP resources hosted by on server.

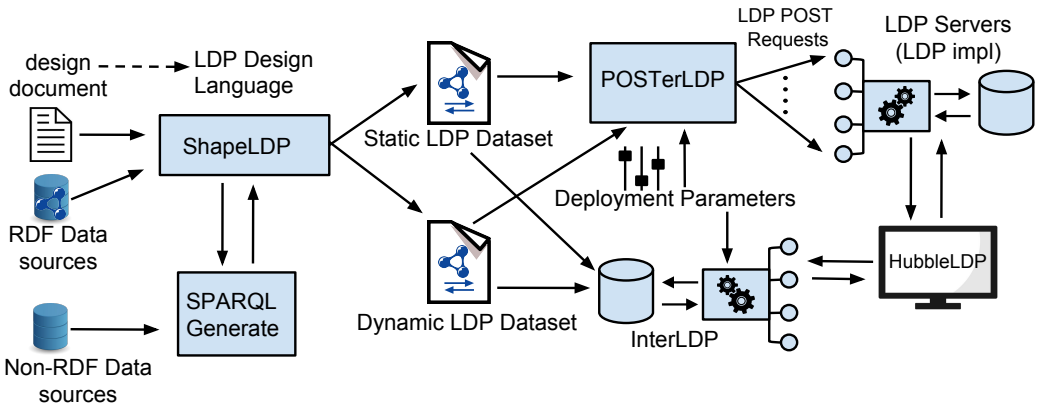

FIGURE 5.1: Overview of LDP Generation Toolkit

The LDP generation workflow can distributed as geographically dispersed actors may implement different part of it using different tools from the toolkit. As we explain in Section 5.3.3, design documents can be modularized. Consequently, different actors (e.g data providers or publishers) may contribute to different part of an LDP design in different documents all of which may be unioned together to form a single design document. For instance, actors standardizing ontologies may provide design in LDP-DL for publishing data structured using these ontologies. Eventually, data publishers may use several of these designs based on the ontologies per which their data is structured, merge them to form a single design document and use it to publish their data using LDPs.

Moreover, as we will see, static/dynamic LDP dataset use relative IRIs making them independent of deployment. Exploiting this feature, different teams at the data publisher's site may be involved in the publication. For example, LDP-DL designers define a design, encode it in LDP-DL and generate the LDP dataset without being aware of deployment aspects (e.g. LDP server address). At deployment time, LDP system administrators may characterize the deployment aspects and deploy the LDP dataset.

Finally, data consumers may use different tools that are completely agnostic of LDP-DL. Moreover, they may ignore the implementation-specific details of the LDPs generated and develop tools considering only the LDP standard. The LDP

browser HubbleLDP is an example of such a tool that may be used for navigating ontologies.

Now, that we have provided an overview of the LDP generation toolkit9 instance, let us proceed with a brief description of the LDP-DL concrete syntax.

5.2 LDP-DL Concrete Syntax

LDP designs document describing the design of LDP in LDP-DL are encoded in RDF graphs that are serialized in documents. The serialization can take place in various RDF syntaxes such as `Turtle`, `RDF/XML`, `JSON-LD`, etc. LDP-DL enables different types of processor's implementations. For example, processors can offer a virtual access to LDP resources over the design documents or generate materialized LDP datasets.

Listing 5.1 shows an example of a design document in the concrete syntax. This design document is in fact the serialization of the LDP-DL design model shown in Chapter 4 (Figure C.1) in the abstract syntax. In the remaining of this section, we describe constructs from the concrete syntax used to write this design document. Our description follows the structure of the document and we make correspondence to constructs from the abstract syntax described in Chapter 4 (Section 4.2.2) wherever possible.

```
1 @prefix : <http://example.com/data/> .
2 @prefix ldl: <https://w3id.org/ldpdl/#> .
3 @prefix dcat: <http://www.w3.org/ns/dcat#> .
4 @prefix dct: <http://purl.org/dc/terms/>.
5
6 # definition of design document and links to its top-level maps
7 <> a ldl:DesignDocument;
8    ldl:topLevelMap :catalog .
9
10 # :catalog describe containers for DCAT catalogs
11 :catalog a ldl:ContainerMap;
12    ldl:resourceMap :rm1;
13    ldl:containerMap :dataset .
14
15 :rm1 a ldl:ResourceMap;
16    # query pattern of ContainerMap :rm1
17    ldl:resourceQuery "{ ?{res} a dcat:Catalog .}";
18
19    # CONSTRUCT query of :rm1
20    ldl:graphQuery "CONSTRUCT { ?{nres} foaf:primaryTopic ?{res} . ?{res}
          ?p ?o . } WHERE { ?{res} ?p ?o . FILTER (?p not in (dcat:dataset))
           }";
21
22    # DataSource query of :rm1
23    ldl:dataSource :dSourceParis .  .
24
25 # :dataset describe containers for DCAT datasets
26 :dataset a ldl:ContainerMap;
27    ldl:slugQueryTemplate "{ BIND (replace(str(?{res}),'http://','') as ?
          oldtemplate) BIND (replace(?oldtemplate,'/','.') as ?{slug}) }";
28    ldl:containerMap :distribs;
29    ldl:resourceMap :rm2 .
30
31 :rm2 a ldl:ResourceMap;
32    ldl:resourceQuery "{ ?{parent} dcat:dataset ?{res} .}";
```

```
33    ldl:graphQuery "CONSTRUCT { ?{nres} foaf:primaryTopic ?{res} . ?{res}
          ?p ?o . } WHERE { ?{res} ?p ?o . FILTER (?p not in (dcat:
          distribution)) }";
34    ldl:dataSource :dSourceParis .
35
36 # :distribs describe containers for grouping DCAT distributions
37 :distribs a ldl:NullContainerMap;
38    ldl:slugTemplate "distributions";
39    ldl:nonContainerMap :distrib;
40    ldl:resourceMap :rm3 .
41
42 :rm3 a ldl:ResourceMap;
43    ldl:graphQuery "CONSTRUCT { ?{nres} dct:description ?title .} WHERE {
          ?{parent} dct:title ?title . BIND(CONCAT('Describes distribution
          of ',?title)) }";
44    ldl:dataSource :dSourceParis .
45
46
47 # :distrib describe non-containers for DCAT distributions
48 :distrib a ldl:NonContainerMap;
49  ldl:resourceMap :rm4 .
50
51
52 :rm4 a ldl:ResourceMap;
53    ldl:resourceQuery "{ ?{parent.parent} dcat:distribution ?{res} .}";
54    ldl:graphQuery "CONSTRUCT { ?{nres} foaf:primaryTopic ?{res} . ?{res}
          ?p ?o . } WHERE { ?{res} ?p ?o . }";
55    ldl:dataSource :dSourceParis .
56
57
58 :dSourceParis a ldl:RDFFileDataSource;
59    ldl:location "https://opendata.paris.fr/api/v2/catalog/exports/ttl" .
```

Listing 5.1: Example of a Design Document in the concrete syntax

At the beginning of the document, we define some prefixes among which `ldl` is the namespace of LDP-DL vocabulary and `:` is an example namespace that instantiate constructs from the vocabulary. Then, there is the definition of the design document and its top-level container maps and non-container maps are specified using `ldl:topLevelMap`. While specifying the top level instances is not obligatory, doing so prevents the processor from having to search for them. Normally, a container map or non-container map is at the top-level if it is not referred by any other container map.

`:catalog` is a instance of `ldl:ContainerMap` that represents a `ContainerMap` and it is at the top-level. Its `ResourceMap` `:rm1` is specified by `ldl:resourceMap`. The query pattern and CONSTRUCT query of the `:rm1` is specified by `ldl:resourceQuery` and `ldl:graphQuery` and all prefixes used in either of them should be defined at the beginning of the document. For example, `dcat:Catalog` is used in the `:rm1`'s query pattern and defined together at the beginning of the document. In the queries, reserved variables are can be identified within `?{...}`. More precisely, `?{res}` is used for the reserved variable ρ and `?{nres}` is used for the reserved variable ν. Also, `:rm1` has a `DataSource` `:dataSource1` specified by `ldl:dataSource`. `:dataSource1` is an instance of `ldl:RDFFileDataSource` which means that it an RDF source whose location is specified by `ldl:location`.

:catalog has a ContainerMap :dataset specified by ldl:containerMap. :dataset has an additional property slugQueryTemplate that is optional for all instances of ldl:ContainerMap and ldl:NonContainerMap. It is used to obtain the value of the header slug, that as we mentioned in Chapter 2 (Section 2.4.2), provide a suggestion to the LDP server for the IRIs of resources. To obtain the value, an LDP-DL processor may evaluate the query pattern on the data source and project the first solution mapping to the reserved variable ?{slug}.

:dataset has a ContainerMap :distribs. As we can see, :distribs is a :NullContainerMap that is a syntax sugar introduced in the concrete syntax. Its aim is to facilitate the definition of ContainerMaps that either do not have ResourceMaps, or have ResourceMaps whose query pattern may not be specified. Containers generated from such ContainerMaps are used for grouping LDP resourcs. For example, the ResourceMap :rm3 of :distribs has no ldl:resourceQuery property. But a generic one will be created for it when processing the design document which is why in the abstract model in Chapter 4 (Figure C.1), :rm3 has a query pattern. Eventually, all ContainerMaps defined using :NullContainerMap are converted to standard instances of ldl:ContainerMap

Also, as we can see, :distribs has a property ldl:slugTemplate. Compared to slugQueryTemplate, slugTemplate directly specifies a value that may be used for the LDP header slug. Finally, :distribs has a NonContainerMap :distrib that is specified by the property ldl:nonContainerMap.

The above description is restricted to the design document in Listing 5.1. For a detailed description of the concrete syntax together with a mapping from the abstract syntax to the concrete one and vice versa, refer to Appendix B.

5.3 LDPizer: ShapeLDP

In this section, we describe ShapeLDP[1], our implementation of the LDPizer (described in Chapter 3, Section 3.3.1). First, we provide an overview of ShapeLDP in Section 5.3.1. Then, we explains how ShapeLDP facilitates the modularization of design document in Section 5.3.3. Finally, in Section 5.3.4, we describe how ShapeLDP generates IRIs for LDP resources.

5.3.1 Overview

ShapeLDP is an open source implementation of an LDPizer programmed in Java on top of the Apache Jena ARQ SPARQL 1.1 engine[2]. It provides a command line interface whose options are shown in Table 5.1.

ShapeLDP consumes a design document, through the designDocument parameter, and evaluate it in either the *static* and *dynamic* based on the mode parameter. We describe static and dynamic evaluation modes in more details in Section 5.3.2 below. The data sources exploited during the evaluation can be in RDF or heterogeneous formats. To exploit heterogeneous data sources, ShapeLDP uses their RDF transformation document (or lifting rules) specified for DataSources in the

[1] https://github.com/noorbakerally/ShapeLDP

[2] https://jena.apache.org/documentation/query/index.html, last accessed 17 July 2018

Short Option	Long Option	Description
d	designDocument	Path of design document
m	mode	Mode of operation that may be static or dynamic
ids	inputDataSource	URL for the main input source
se	sparqlEndpoint	URL for a SPARQL endpoint as main input source
lf	liftingRule	Lifting rule for the main input source
sd	sourceDocument	Source documents containing partial design definitions
o	ouputFile	Path to output LDP dataset
l	logging	Disable logging by setting value to true or false
h	help	Show Help

TABLE 5.1: ShapeLDP command line options

design model. For now, ShapeLDP can only consume lifting rules written in SPARQL-Generate [LZB17a] are supported. Future versions may consider other languages such as RML [DVSC+14], XSPARQL [AKKP08a] or others.

During the design document evaluation, if a single data source is used, its URL can be passed to ShapeLDP via the `inputDataSource` or `sparqlEndpoint` parameter in case the source is an (Non-)RDF source or SPARQL endpoint respectively. These parameters are mutually-exclusive and if they are used on a design document containing already a definition of data sources, such as the one in Listing 5.1, these definition are overrides and only the one specified by either of these parameters is considered. Also, if the data source specified by `inputDataSource` is not in RDF, its transformation document may be passed through the `liftingRule` parameter.

Moreover, as we explain in Section 5.3.3 below, it is possible to modularize an LDP-DL design in several documents that may then be input using the `sourceDocument`. Finally, the path of the output LDP dataset can be specified using the `outputFile` parameter.

5.3.2 Design Document Processing

ShapeLDP can process design documents in either the static or dynamic mode that are described in Section 5.3.2 and Section 5.3.2 respectively.

Static Mode

In the static mode, ShapeLDP generates a static LDP dataset. As mentioned in Chapter 4 (Section 4.3.2), static LDP dataset uses temporary identifiers. In our implementation, we use relative IRIs as these temporary identifiers. An example of a static LDP dataset is shown in Listing 5.2.

```
1 @prefix ldl: <https://w3id.org/ldpdl/ns#> .
2 @prefix bcat: <https://bistrotdepays.opendatasoft.com/api/v2/catalog/
      exports/> .
3 @prefix bdataset: <https://bistrotdepays.opendatasoft.com/api/v2/catalog/
      datasets/> .
4 @prefix ldp: <http://www.w3.org/ns/ldp#> .
5 @prefix foaf: <http://xmlns.com/foaf/0.1/>.
6 @prefix dcat: <http://www.w3.org/ns/dcat#>.
```

```
 7 @prefix dc: <http://purl.org/dc/terms/>.
 8
 9 # definition of static LDP dataset & metadata in the default graph
10 <> a ldl:StaticLDPDataset;
11   # metadata include top-level resources, all (non-)containers
12   ldl:topLevelResource <catalog>;
13   ldl:container <catalog>;
14   ldl:nonContainer <catalog/animations> .
15
16 # metadata can also include members of all containers
17 <catalog> ldp:contains <catalog/animations> .
18
19 # description of LDP resources
20 <catalog> {
21   <catalog> a ldp:BasicContainer;
22     ldp:contains <animations> ;
23     foaf:primaryTopic bcat:ttl .
24
25   bcat:ttl a dcat:Catalog;
26     dc:description "Bistrotdepays Catalog" .
27 }
28
29 <catalog/animations> {
30   <catalog/animations> a ldp:RDFSource;
31     foaf:primaryTopic bdataset:animations_bistrots_de_pays .
32     bdataset:animations_bistrots_de_pays a dcat:Dataset;
33     dc:description "Liste des animations dans les etablissements du label
         Bistrot de Pays" ;
34     dc:title "Animations Bistrots de Pays" .
35 }
```

Listing 5.2: Example of a static LDP dataset

Let us now describe the above static LDP dataset that was generated by ShapeLDP. As we can see, ShapeLDP serializes static LDP datasets as RDF datasets in the TriG format [CS14b]. To facilitate the processing of static LDP dataset, it adds some metadata in its default graph using the vocabulary that is described in Appendix B.1. For example, the document is typed an `ldl:StaticLDPDataset` (Listing 5.2, Line 10) because as mentioned before, there are also dynamic LDP datasets. Moreover, top-level resources, whether containers or non-containers, are specified using the `ldl:topLevelResource` such as `<catalog>` (Line 12). Also, all containers and non-containers are specified using `ldl:container` and `ldl:nonContainer` respectively such as `<catalog>` (Line 13) and `<catalog/animations>` (Line 14). Finally, all the members of containers are always specified. In this case, only the members of `<catalog>` (Line 17) are specified because it is the only container. In Section 5.4, we explain how the metadata may be used when deploying LDP datasets.

In Chapter 4 (Section 4.3.2), we have describe the formalization of static LDP datasets. Using the description, when serializing a non-container (n, g), a named graph is created with the relative IRI n and graph g. With containers (n, g, M), the procedure is different due to the existence of members. For every container, we create a named graph with the relative IRI n and its serialized graph being the union of g and triples for every member from M with subject, predicate and object of a triple being n, `ldp:contains` and $m \in M$ respectively. For example, both `<catalog>` (Line 20) and `<catalog/animations>` (Line 34) are container and

non-container respectively and have their corresponding graph. Also, the graph of <catalog> uses triples with predicates ldp:contains to specify its members.

Dynamic Mode

In the dynamic mode, ShapeLDP generates dynamic LDP datasets. As in the case of static mode, relative IRIs are generated as temporary identifiers for LDP resources. An example of a dynamic LDP dataset is shown in Listing 5.2.

```
@prefix ldl: <https://w3id.org/ldpdl/ns#> .
@prefix bcat: <https://bistrotdepays.opendatasoft.com/api/v2/catalog/
    exports/> .
@prefix bdataset: <https://bistrotdepays.opendatasoft.com/api/v2/catalog/
    datasets/> .
@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix dcat: <http://www.w3.org/ns/dcat#>.
@prefix dc: <http://purl.org/dc/terms/>.

# definition of dynamic LDP dataset & metadata in the default graph
<> a ldl:DynamicLDPDataset;
  ldl:topLevelResource <catalog>;
  ldl:container <catalog>;
  ldl:nonContainer <catalog/animations> .
<catalog> ldp:contains <catalog/animations> .

# description of LDP resources
<catalog> {
  <catalog> a ldp:BasicContainer;
    ldp:contains <animations> .

  # graph description provides compiled CONSTRUCT query
  # and data source on which query must be executed
  <catalog> ldl:graphDescription [
    ldl:graphQuery "CONSTRUCT { <https://bistrotdepays.opendatasoft.com/
        api/v2/catalog/exports/ttl> ?p ?o . } WHERE { <https://
        bistrotdepays.opendatasoft.com/api/v2/catalog/exports/ttl> ?p ?o
        FILTER ( ?p NOT IN (<http://www.w3.org/ns/dcat#dataset>) ) } " ;
    ldl:dataSource <DataSource1>
  ] .



  <DataSource1> a ldl:DataSource>;
    ldl:location "http://bistrotdepays.opendatasoft.com/api/v2/catalog/
        exports/ttl" .

}
<catalog/animations> {
  <catalog/animations> a ldp:RDFSource .

  <catalog/animations> ldl:graphDescription [
    ldl:graphQuery "CONSTRUCT { <https://bistrotdepays.opendatasoft.com/
        api/v2/catalog/datasets/animations_bistrots_de_pays> ?p ?o . }
        WHERE { <https://bistrotdepays.opendatasoft.com/api/v2/catalog/
        datasets/animations_bistrots_de_pays> ?p ?o FILTER ( ?p NOT IN (<
        http://www.w3.org/ns/dcat#distribution>) ) } " ;
    ldl:dataSource <DataSource1>
  ] .

  <DataSource1> a ldl:DataSource>;
```

```
43    ldl:location "http://bistrotdepays.opendatasoft.com/api/v2/catalog/
          exports/ttl" .
44
45 }
```

Listing 5.3: Example of a dynamic LDP dataset

This dynamic LDP dataset was generated from the source and using the same design document as the static LDP dataset in Listing 5.2 which explains why it has the same metadata with the exception that it is typed as an `ldl:DynamicLDPDataset` (Line 10).

In Chapter 4 (Definition 18), we have describe the formalization of static LDP datasets. Using the description, when serializing dynamic containers or non-containers, a named graph is created for them with their relative IRIs, `GraphDescription` are serialized in the named graph. Additionally, if they are containers, their named graph would include triples where the subjects are the containers' relative IRIs, the predicate are `ldp:contains` and the objects are the members' relative IRI. For example, in both the container `<catalog>` and non-container `<catalog/animations>`,their `GraphDescription` is specified by `ldl:graphDescription`. Moreover, for `<catalog>`, it only member is specified using `ldp:contains` in its named graph.

5.3.3 Modularizing Design Document

As mentioned before, the concrete syntax of LDP-DL is in RDF and the definition of an LDP-DL design involves the definition of several constructs namely `ContainerMaps`, `NonContainerMaps`, `ResourceMaps` and `DataSources`. ShapeLDP takes advantage of the RDF data model and enhances modularization by allowing design documents to be split into different files that can contain partial definition of an LDP-DL design. To process these files, ShapeLDP consumes their URLs and combine them into a single LDP-DL design model before interpreting it. Modularizing a design design enhances the reusability of the partial designs.

For example, the design document given in Listing 5.1 include the definition of a `DataSource` (Listing 5.1, Line 23 - 59). The design document can be be modularized into two different documents, one containing the main LDP-DL design, shown in Listing 5.4, and one containing the definition of the data source, shown in Listing 5.5.

```
1 @prefix ldl: <https://w3id.org/ldpdl/#> .
2 @prefix : <http://example.com/data/> .
3 @prefix dcat: <http://www.w3.org/ns/dcat#> .
4 @prefix dct: <http://purl.org/dc/terms/>.
5
6 # definition of design document & metadata
7 <> a ldl:DesignDocument;
8   ldl:topLevelMap :catalog .
9
10 # :catalog describe containers for DCAT catalogs
11 :catalog a ldl:ContainerMap;
12   ldl:resourceMap :rm1;
13   ldl:containerMap :dataset .
14
15 # :dataset describe containers for DCAT datasets
16 :dataset a ldl:ContainerMap;
```

```
17  ldl:slugQueryTemplate "{ BIND (replace(str(?{res}),'http://','') as ?
        oldtemplate) BIND (replace(?oldtemplate,'/','.') as ?{slug}) }";
18  ldl:containerMap :distribs;
19  ldl:resourceMap :rm2 .
20
21 # :distribs describe containers for grouping DCAT distributions
22 :distribs a ldl:NullContainerMap;
23  ldl:slugTemplate "distributions";
24  ldl:nonContainerMap :distrib;
25  ldl:resourceMap :rm3 .
26
27 # :distrib describe containers for DCAT distributions
28 :distrib a ldl:NonContainerMap;
29  ldl:resourceMap :rm4 .
30
31 # Definition of ResourceMaps without their data sources
32 :rm1 a ldl:ResourceMap;
33  ldl:resourceQuery "{ ?{res} a dcat:Catalog .}";
34  ldl:graphQuery "CONSTRUCT { ?{nres} foaf:primaryTopic ?{res} . ?{res} ?p
         ?o . } WHERE { ?{res} ?p ?o . FILTER (?p not in (dcat:dataset)) }" .

35
36 :rm2 a ldl:ResourceMap;
37  ldl:resourceQuery "{ ?{parent} dcat:dataset ?{res} .}";
38  ldl:graphQuery "CONSTRUCT { ?{nres} foaf:primaryTopic ?{res} . ?{res} ?p
         ?o . } WHERE { ?{res} ?p ?o . FILTER (?p not in (dcat:distribution)
        ) }" .
39
40 :rm3 a ldl:ResourceMap;
41  ldl:graphQuery "CONSTRUCT { ?{nres} dct:description ?title .} WHERE { ?{
        parent} dct:title ?title . BIND(CONCAT('Describes distribution of
        ',?title)) }" .
42
43 :rm4 a ldl:ResourceMap;
44  ldl:resourceQuery "{ ?{parent.parent} dcat:distribution ?{res} .}";
45  ldl:graphQuery "CONSTRUCT { ?{nres} foaf:primaryTopic ?{res} . ?{res} ?p
         ?o . } WHERE { ?{res} ?p ?o . }" .
```

Listing 5.4: Document containing partial LDP-DL design without data source definitions

```
1 @prefix on: <https://w3id.org/ldpdl/#> .
2 @prefix : <http://example.com/data/> .
3
4 # linking resource maps to Paris DCAT catalog data source
5 :rm1 a on:dataSource :dataSource1 .
6 :rm2 a on:dataSource :dataSource1 .
7 :rm3 a on:dataSource :dataSource1 .
8 :rm4 a on:dataSource :dataSource1 .
9
10 # definition of Paris DCAT catalog data source
11 :dataSource1 a on:RDFFileDataSource;
12  on:location "https://opendata.paris.fr/api/v2/catalog/exports/ttl" .
```

Listing 5.5: Document containing data source definitions for Paris DCAT catalog

Modularizing the design documents enhances their reusability. For example, the partial LDP-DL design in Listing 5.4 may be used to generate LDPs from any other data sources by using it in combination with their definitions such as the one shown in Listing 5.6 or the one in Listing 5.6 Listing 5.6.

```
@prefix ldl: <https://w3id.org/ldpdl/#> .
@prefix : <http://example.com/data/> .

# linking resource maps to toulouse data source
:rm1 a ldl:dataSource :dSourceToulouse .
:rm2 a ldl:dataSource :dSourceToulouse .
:rm3 a ldl:dataSource :dSourceToulouse .
:rm4 a ldl:dataSource :dSourceToulouse .

# definition of toulouse data source
:dSourceToulouse a ldl:RDFFileDataSource;
  ldl:location "https://data.toulouse-metropole.fr/api/v2/catalog/exports/
      ttl" .

# linking resource maps to nantes data source
:rm1 a ldl:dataSource :dSourceNantes .
:rm2 a ldl:dataSource :dSourceNantes .
:rm3 a ldl:dataSource :dSourceNantes .
:rm4 a ldl:dataSource :dSourceNantes .

# definition of nantes data source
:dataSource3 a ldl:RDFFileDataSource;
  ldl:location "https://data.nantesmetropole.fr/api/v2/catalog/exports/ttl
      " .
```

Listing 5.6: Document containing data source definitions for Toulouse DCAT catalog

When design documents are modularized, ShapeLDP merges RDF graphs from the documents to obtain to obtain a single design model. For example, merging the RDF graphs from the documents in Listing 5.5 and Listing 5.6 would result in a single design model that is then processed by ShapeLDP.

The latter example used is a concrete case where the modularizing design documents may enhance their reusability. By separating the definition of the `DataSources` in a different file, the reusability of the design in Listing 5.1 increases as anyone may have their own definition of data sources in a different document and use it together with the latter document as it is. There may be other reusability patterns. For example, the queries specified by `ldl:resourceQuery` and `ldl:graphQuery` could be modularized in a different documents to abstract the main LDP-DL design from the varying vocabularies used in data sources. Then, document containing queries specific to a particular data source may be used in combination with the document containing the main LDP-DL design.

5.3.4 LDP Resource IRI Generation

As mentioned before, both static and dynamic LDP datasets use relative IRIs. In their families of algorithms, we have assumed the existence of a function $genIRI$ that generates these relative IRIs but have not explicitly described it. In ShapeLDP, the implementation of this function exploits the `slugQueryTemplate` or `slugTemplate`, described in Section 5.2, to obtain a text that it uses to create the relative IRI.

Let us consider an example to describe how ShapeLDP generates relative IRIs. Suppose when creating the IRI of an LDP resource, it obtains the text “paris”. If the LDP resource is not contained in any container, i.e. a top-level resource, then that

text is its final relative IRI. However, if the LDP resource is a member of a container having the relative IRI "france/city", then its relative IRI will be "france/city/paris" that is obtained by concatenating the text obtained for LDP resource with the relative IRI of its container. If duplicate relative IRIs are generated, then random numbers are concatenated until they are unique.

5.4 LDP Dataset Deployer: POSTerLDP

In this section, we describe POSTerLDP, our implementation of the LDP dataset deployer (described in Chapter 3, Section 3.3.2). First, we provide an overview of POSTerLDP in Section 5.4.1. Then in Section 5.4.2 and Section 5.4.3, we describe the write and update modes respectively in which POSTerLDP can operate.

5.4.1 Overview

POSTerLDP[1] is open source and implemented in Python on top of the RDF library RDFLib[2]. It is completely agnostic of the LDP server's implementation. It generates standard LDP requests and is thus compatible with any server implementing LDP interactions. Currently, POSTerLDP can deploy LDP resources on only one server but future versions may consider replication or partitioning schemes described in a particular deployment language.

Option	Description
baseURL	Base URL of the LDP server
username	Username for basic authentication
password	Password for basic authentication
LDPdataset	URL of LDP dataset
staticLDPdataset	URL of static LDP dataset
dynamicLDPdataset	URL of dynamic LDP dataset
empty	Generate empty RDF graph for LDP resources
mode	POSTerLDP mode, either write or update
help	Show help message

Table 5.2: POSTerLDP command line options

Table 5.2 shows a list of options provided by the command line interface of POSTerLDP. Basically, `baseURL` corresponds to the base URL on the LDP where resources are deployed and `username` and `password` are parameters used for basic authentication. Also, POSTerLDP can directly consume LDP datasets as well as static/-dynamic LDP datasets and `LDPdataset`, `staticLDPdataset` and `dynamicLDPdataset` are mutually exclusive parameters for passing the required document.

In case a static LDP dataset is consumed, the LDP dataset is instantiated from it by resolving all relative IRIs for containers and non-containers to absolute IRIs using the base URL of the LDP server. When consuming a dynamic LDP dataset,

[1] https://github.com/noorbakerally/POSTerLDP, last accessed on 10 July 2018
[2] http://rdflib.readthedocs.io/en/stable/, last accessed on 17 July 2018

POSTerLDP generates an LDP dataset using Algorithm 11 where the RDF graphs of LDP resources are fixed. POSTerLDP may then deploy the latter LDP dataset.

When deploying using POSTerLDP, due to the dynamicity of data sources, the latter LDP dataset may become invalid. To cater for this issue, we provide an additional option `empty` via the command interface that allows generating LDP dataset where the RDF graphs of LDP resources are empty. In this way, the latter LDP dataset may be deployed on the LDP server and extensions may be made to the server by the party hosting the LDP so that it uses the dynamic LDP datasets to generate the RDF graphs of the LDP resources.

Finally, the parameter `mode` can either be *write* or *update*. These two modes are further described in the next two sections.

5.4.2 Write Mode

In the write mode, POSTerLDP deploys all resources described in the LDP dataset using LDP `POST` requests with the assumption that the LDP server does not already contain any resources described in the LDP dataset. The deployment of one resource is realized by $WriteResource$ (cf. Algorithm 5) that takes as parameters the LDP resource to deploy (res), the location of the pre-existing LDP container where res is to be deployed ($location$), and credentials ($username$ and $password$), if any, for basic authentication on the server.

$WriteResource$ only performs the deployment of one resource and an LDP dataset may contain a number of resources. It is not possible to simply iterate on all resources from the LDP dataset and deploy them using $WriteResource$. This is because, there may be resources that are actually members of a particular container and when deploying resources using `POST`, the containers have to be deployed first before their members. Therefore, to deploy an LDP dataset, first its top-level resources are deployed then their members are recursively deployed if they are containers.

Algorithm 5 Deployment of an LDP resource in Write mode

```
1:  procedure WriteResource(res, location, username, password)
2:      //create POST request
3:      postReq ← GeneratePostReq(res, location, username, password)
4:      postResp ← SendPostReq(postReq)
5:      if StatusCode(postResp) = 201 then //check the status code
6:          //resource was created
7:          if res is a Container then
8:              contLoc ← Header(postResp, "Location") //retrieve Location header
9:              for all memRes ∈ members(res) do //create members in container
10:                 WriteResource(memRes, contLoc, username, password)
11:             end for
12:         end if
13:     else
14:         generateError(postReq, postResp)
15:     end if
16: end procedure
```

Let us now briefly describe $WriteResource$. To deploy the resource res, first a conformant LDP `POST` request is generated using the four parameters (Algorithm 5,

Line 3). This involves setting the request body and the proper header per the LDP standard. Once created, the POST request is sent and the reply $postResp$ is obtained. The status of the response is checked (Algorithm 5, Line 5) and a status code of 201 signifies that the resource has successfully been created [SAM15c, §5.2.3.2]. If it is the case and if res is a container, then all its members are retrieved and are deployed where it has been deployed on the server (Algorithm 5, Line 7 - 12). Per the LDP standard, when creating a resource through POST, its deployment location is returned in the Location header of the response [SAM15c, §5.2.3.2]. Therefore, the deployment location header of resource is retrieved and stored in $contLoc$ (Algorithm 5, Line 8) and used when sending POST request for its members.

In POSTerLDP, $WriteResource$ exploits the fact that ShapeLDP places direct references to top-level resources in the default graph of the serialized LDP dataset as described in Section 5.3.2. Thus, when deploying an LDP dataset, POSTerLDP looks for these references and starts its deployment from there. However, if the default graph is empty, then it assumes that the static/dynamic LDP dataset has not been generated by ShapeLDP. Consequently, it naively identifies the top-level resources by iterating through all containers and non-containers in the LDP dataset and checking whether they are members of any container. Finally, it deploys the top-level resources and recursively deploy their members if they are containers.

5.4.3 Update Mode

In the update mode, POSTerLDP assumes that the server on which the LDP dataset is to be deployed may contain LDP resources from the LDP dataset. $updateRes$ (Algorithm 6) shows the deployment of a particular LDP resource in this mode. It takes exactly the same parameters as $WriteResource$.

When deploying a resource res, first, POSTerLDP checks whether it already exist on the server by sending a HEAD request using its IRI (Algorithm 6, Line 3). If a response with a status code of 200 is received, meaning that the resource exists, it is updated through a PUT request using its new content from the LDP dataset (Algorithm 6, Line 4). Otherwise, if the status code is 404, meaning that it does not exist on the server, it is created using a POST request (Algorithm 6, Line 11 - 13). Finally, if res is a container, its members are recursively updated (Algorithm 6, Line 17).

Like $WriteResource$ (Algorithm 5), $UpdateResource$ also describes the deployment on one LDP resource and therefore requires identification of top-level resources and the recursive deployment of their members if they are containers.

In summary, POSTerLDP deploys an LDP dataset by creating and/or modifying resources on LDP server using the LDP write interaction model. Another way is to use an LDP server, described in the next section, that can directly instantiate an LDP using an LDP dataset.

Algorithm 6 Deployment of an LDP resource in update Mode

procedure $UpdateResource(res, location, username, password)$
  $headReq \leftarrow GenerateHeadReq(res)$//create head request
  $headResp \leftarrow SendHeadReq(res, dep)$
  if $StatusCode(headResp) = 200$ **then** //check if resource exists
    //Update the resource
    $putReq \leftarrow GeneratePutReq(res)$
    $putResp \leftarrow SendPutReq(res, dep)$
    $resIRI \leftarrow iri(res)$
  else if $StatusCode(headResp) = 404$ **then** //check if the resource does not exists
    //create the resource
    $postReq \leftarrow GeneratePostReq(res, location, username, password)$
    $postResp \leftarrow SendPostReq(postReq)$
    $resIRI \leftarrow Header(postResp, \text{“}Location\text{”})$
  end if
  if res is a Container **then**
    for all $memRes \in members(res)$ **do**//update members of container res
      $UpdateRes(memRes, resIRI, username, password)$
    end for
  end if
end procedure

5.5 LDP Server: InterLDP

InterLDP[1] is an LDP server that can directly consume an LDP dataset (static or dynamic) and exposes LDP resources from it. As of now, it only supports HTTP GET, HEAD and OPTIONS requests LDP-RSs.

InterLDP can functions in either the static and dynamic modes that are described in Section 5.5.1 and Section 5.5.2 respectively.

5.5.1 Static Mode

In this mode, InterLDP takes as input a static LDP dataset or directly an LDP dataset and the base URL at which the LDP server is going to be exposed. If it consumes a static LDP dataset, then it instantiate the LDP dataset from it by resolving all relative IRIs for containers and non-containers to absolute IRIs using the base URL. $EvalGet$ (Algorithm 7) describes how InterLDP evaluates `GET` requests on LDP resources in the static mode on the LDP dataset Σ.

Algorithm 7 Evaluation of `GET` request by InterLDP in Static Mode

procedure $EvalGet(req, \Sigma)$
  $res \leftarrow getResource(\Sigma, iri(req))$ //get the requested resource
  if $res = \varnothing$ **then**
    $response \leftarrow generateError(req)$
  else
    $response \leftarrow generateReply(res)$ //create response with required headers & body
  end if
  return $response$
end procedure

[1] https://github.com/noorbakerally/InterLDP

On obtaining a request for a resource, first the resource is retrieved from Σ. For this, we assume the existence of a function $getResource(\Sigma, resIRI)$ that retrieves a resource with IRI $resIRI$ from the LDP dataset Σ (Algorithm 7, Line 2). If Σ does not contain such a resource, an error is generated with the appropriate content per the LDP standard (Algorithm 7, Line 4). Otherwise, using the resource retrieved, a response is generated by setting the appropriate response body and headers per the LDP standard. Finally, the response is returned (Algorithm 7, Line 6).

5.5.2 Dynamic Mode

In this mode, InterLDP takes as input a dynamic LDP dataset and the base URL at which the LDP server is going to be exposed. $EvalGet$ (Algorithm 8) describes how InterLDP evaluates `GET` requests on LDP resources in the dynamic mode the dynamic LDP dataset Δ.

Algorithm 8 Evaluation of `GET` request by InterLDP in Dynamic Mode

1: **procedure** $EvalGet(req, \Delta)$
2: $res \leftarrow getDynamicRes(\Delta, iri(req))$ //get the dynamic container or non-container
3: **if** $res = \varnothing$ **then**
4: $response \leftarrow genErrorReply(req)$
5: **else**
6: $gdes \leftarrow GDES(res)$ //retrieve its graph description
7: $g \leftarrow Eval_{gdes}(gdes)$ //generate the graph using its graph description
8: $response \leftarrow generateReply(iri(res), g)$
9: **end if**
10: **end procedure**

On obtaining a request for a resource, it is retrieved from Δ using the function $getResource(\Delta, resIRI)$ (Algorithm 8, Line 2) whose existence we assume. If Δ does not contain the resource, an error is generated (Algorithm 8, Line 4). Otherwise, using the resource res from Δ, its `GraphDescription` is retrieved (Algorithm 8, Line 7) and evaluated using $Eval_{gdes}$(Algorithm 8, Line 6) to generate the graph of the resource. $Eval_{gdes}$, as mentioned before, performs the RDF merge of all the individual `DataSources` in DS to generate an RDF graph on which cq is evaluated to generate the graph of the resource. Finally, a response is generated by setting the appropriate response body and headers per the LDP standard (Algorithm 8, Line 8).

5.6 LDP Browser: HubbleLDP

HubbleLDP[1] is an LDP browser that can be used to browse resources on an LDP and view their content. Figure 5.2 shows a screenshot which is actually an instance[2] of it running loaded with an LDP[3] about DCAT catalogues[4] and organization of its datasets in different languages.

[1] https://github.com/noorbakerally/HubbleLDP
[2] http://opensensingcity.emse.fr/ldp-browser/, last accessed on 30 August 2018
[3] http://opensensingcity.emse.fr/ldpdfend/tourism62/d3/catalog
[4] https://tourisme62.opendatasoft.com/api/v2/catalog/exports/ttl

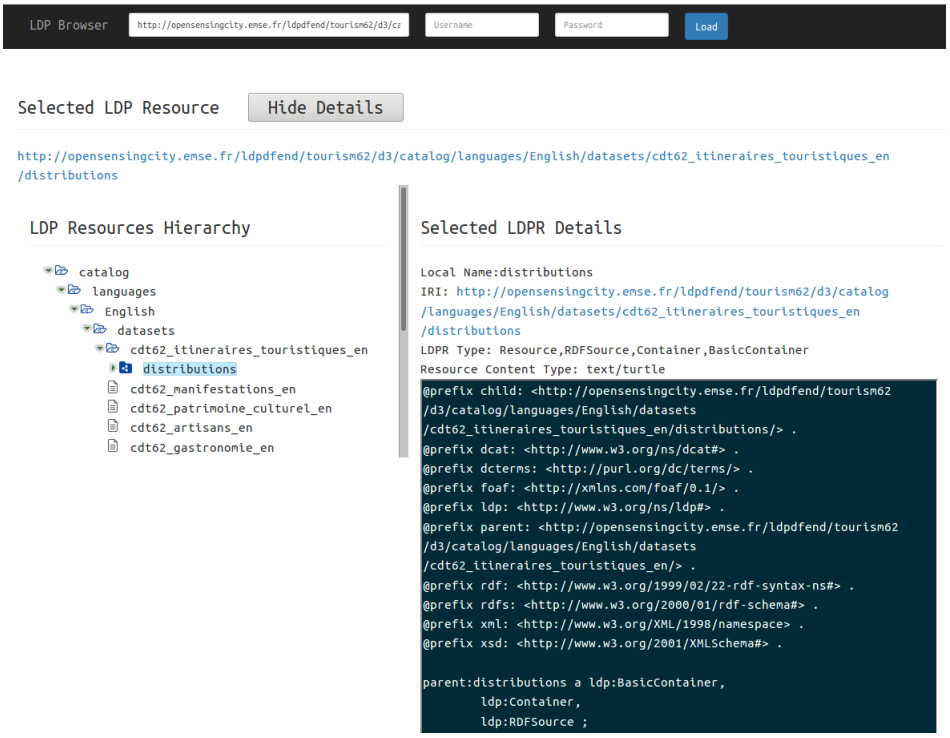

FIGURE 5.2: Screenshot of HubbleLDP

The interface has three main parts, the top part where details of the LDP server are entered, the *LDP Resources Hierarchy* part that display the hierarchy of LDP resources in a directory tree structure and the *Selected LDPR Details* part where details of the selected resource from the LDP Resources Hierarchy is shown.

5.7 Summary

In this chapter, we describe the LDP generation toolkit that is an implementation of the tools in the LDP generation workflow. We provide an overview of the concrete syntax of LDP-DL that is the core of the workflow. Then, we describe ShapeLDP, our implementation of an LDPizer, and more specifically, we describe how it evaluates design documents and generate IRIs for containers and non-containers. We describe POSTerLDP, our implementation of an LDP dataset deployer, and describes the different modes in which it can deploy an LDP dataset on a server. Also, we describe InterLDP, our implementation of an LDP server, that can directly consume static and dynamic LDP dataset and expose them via HTTP. Finally, we provide HubbleLDP, our implementation of an LDP browser, and briefly explain how it can be used to navigate through LDP resources on a particular server.

Chapter 6

Evaluation

In the previous chapter, we described our implementation of the LDP generation workflow consisting of different tools to automatize the generation of LDPs. In this chapter, our aim is to evaluate these tools with respect to the criteria defined in the Introduction of this thesis by doing several experiments with different objectives and constraints.

To this end, first we describe an experiment to analyze the performance of ShapeLDP in Section 6.1. Then, in Section 6.2, we describe several experiments done with respect to our evaluation criteria outlined in the introduction of this thesis. Finally, in Section 6.3, we describe several experiments and show some side contributions of our approach in general.

Contents

6.1 Performance of ShapeLDP

In this section, we describe a performance test on ShapeLDP whose aim is to analyze the time taken by ShapeLDP to execute a design document with respect to RDF graphs fo different sizes. We chose to do this on ShapeLDP only because it is a core of our approach. The processing done by the remaining tools (POSTerLDP, InterLDP, HubbleLDP) are rather straightforward and also a performance test for them may

not be very indicative as their real use may be highly influenced by network aspects (bandwidth, congestion, etc.).

We use ShapeLDP and the design document in Listing 6.1 with respect to RDF graphs of different sizes. In the remaining of this section, first, we briefly describe the generation of the RDF graphs we used in the performance test. Then, in Section 6.1.2, we describe and comment on the results of the performance test.

6.1.1 RDF Graph Generation

We generate RDF graphs of increasing sizes structured per the DCAT vocabulary. We generate RDF graphs structured per the DCAT vocabulary. To do so, an RDF graph having a DCAT catalog is created. Then, incrementally add random DCAT datasets and distributions are added to the DCAT catalog until the number of triples exceeds one million. Random DCAT datasets and distributions are generated by creating random unique IRIs for them. Also, during the generation of the latter RDF graph, at regular intervals, the RDF graph generated so far is serialized. In this way, we obtain several RDF graphs with the biggest one having around one million triple. A more detailed description of the RDF graphs generation together with the program that does it is provided in Appendix D.1.

6.1.2 Test Results

We automate the execution of the first design document from *Domain Design Reusability* (Section 6.2.1) with respect to the 541 RDF graphs generated. We find that the execution time is approximately linear.

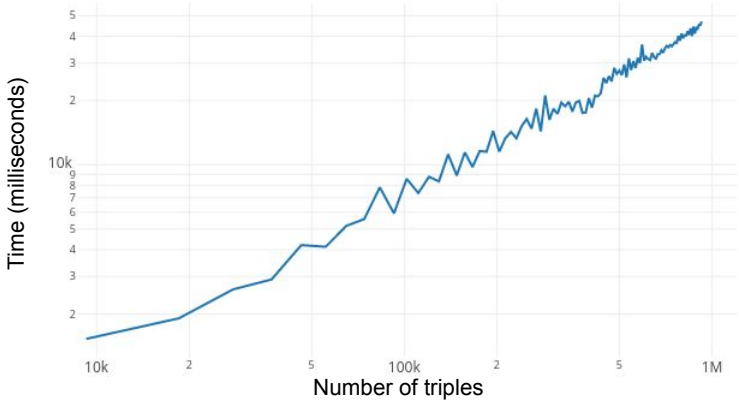

FIGURE 6.1: Execution time of ShapeLDP

In the performance test, we have used an existing design document without much consideration on the aspects of LDP-DL that it uses. The aim was only to have an indication of the performance of ShapeLDP on processing a particular design document on data sources of varying sizes. To perform a full-fledged performance test, though not the aim of this work ,LDP-DL aspects that affect performance need to be identified. For example, since SPARQL is a core element of LDP-DL, SPARQL queries of varying complexity may affect the performance differently. Once these aspects are identified, different types of design documents that uses them may be produced to perform insightful performance tests.

6.2 Evaluation with respect to criteria

In this section, we describe families of experiments. The aim of these experiments is to demonstrate a feature of either LDP-DL, our implementation or our approach of generating LDPs in general with respect to the evaluation criteria outlined in the Introduction of this thesis. For each experiment, we describe the data sources, design documents, generation and deployment of the LDP dataset. These families of experiments concern thus hosting constraints, data heterogeneity, design reusability and automated LDP generation. In each of the experiment belonging to a family and related to an evaluation criteria, we first describe setting parameters and then dicuss the results. Figures use boldface fonts or dotted line shapes to emphasis on particular aspects.

6.2.1 Design Reusability

Design reusability with respect to LDP-DL means that the same design document or part of it is reused independently of the data source or the LDP generated from them. We evaluate this aspect through three experiments. First, we perform an experiment in Section 6.2.1 where we reuse the same design document on different data sources in a particular domain. Then, in Section 6.2.1, we use a generic design document that is independent of any domain and may be reused on data sources that uses the RDFS/OWL vocabulary. Finally, in Section 6.2.1, we demonstrate how part of design documents may be reused through the modularization feature of ShapeLDP.

Domain Design Reusability

In this experiment, we reuse the same design documents on different data sources to generate LDPs. A general overview of the latter experiment is shown in Figure 6.2.

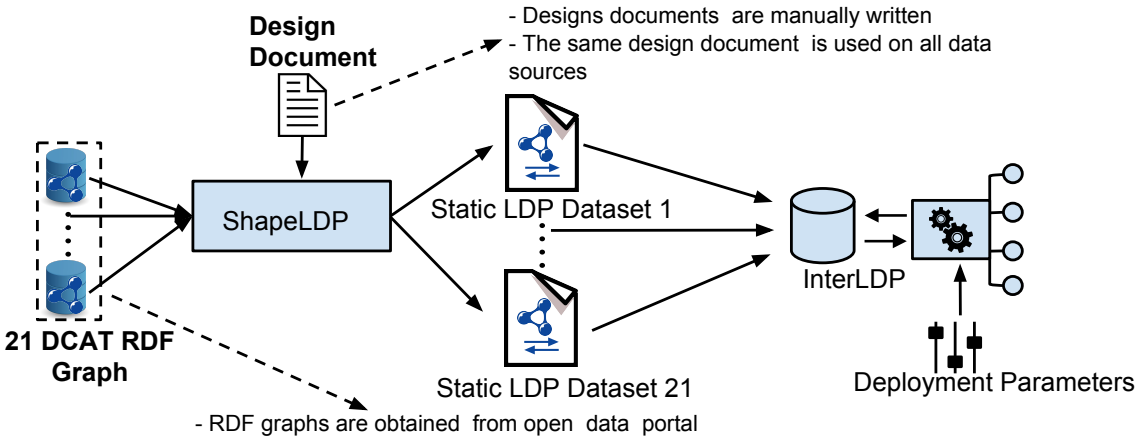

FIGURE 6.2: Overview of *Domain Design Reusability* experiment

We use 21 RDF graphs as input data sources that are structured per the DCAT vocabulary [ME14b]. These RDF graphs are obtained from the list of open data

portals curated by OpenDataSoft[1] that provide a description of their catalogs using the DCAT vocabulary.

The design document we use is shown in Listing 6.1. `:datasetCM` define containers for describing DCAT catalogues that in turn contains the `ContainerMap` `:datasetCM` containers for describing DCAT datasts. Finally, `:datasetCM` contains the `NonContainerMap` `:distributionCM` for describing DCAT distributions. As we see, the design document does not have any explicit data source definition as they are passed to ShapeLDP through the command line option `inputDataSource` described in Chapter 5 (Section 5.3.1) .

```
@prefix ldl: <https://w3id.org/ldpdl/#> .
@prefix :<http://opensensingcity.emse.fr/LDPDesign/data/> .
@prefix dcat: <http://www.w3.org/ns/dcat#>

# :catalogCM describes containers for DCAT catalogs
:catalogCM a ldl:ContainerMap;
   ldl:resourceMap :catalogRM;
   ldl:containerMap :datasetCM .

:catalogRM a ldl:ResourceMap;
  ldl:resourceQuery "{ ?{res} a dcat:Catalog .}";
  ldl:graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o .
      FILTER (?p not in (dcat:dataset)) }" .

# :datasetCM describes containers for DCAT datasets
:datasetCM a ldl:ContainerMap;
   ldl:nonContainerMap :distributionNM;
   ldl:resourceMap :datasetRM .

:datasetRM a ldl:ResourceMap;
  ldl:resourceQuery "{ ?{parent.parent} dcat:dataset ?{res} .}";
  ldl:graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o .
      FILTER (?p not in (dcat:distribution)) }" .

# :distributionNM describes non-containers for DCAT distributions
:distributionNM a ldl:NonContainerMap;
   ldl:resourceMap :distributionRM .

:distributionRM a ldl:ResourceMap;
   ldl:resourceQuery "{ ?{parent.parent} dcat:distribution ?{res} .}";
   ldl:graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o . }"
       .
```

Listing 6.1: First design document in *Domain Design Reusability*

In all, we obtain 21 RDF graphs that we use as data sources to generate 21 static LDP datasets that are used by InterLDP to instantiate different LDPs. In Appendix D.5.2, we provide a the URLs of the RDF graphs.

In this experiment, we have shown the design reusability by generating LDPs from different data sources using the same design document. This was possible because the data sources considered were all structured per the same ontology on which the design document was based on.

[1] https://www.opendatasoft.com/a-comprehensive-list-of-all-open-data-portals-around-the-world/, last accessed on 5 January 2018

Generic Design Reusability

In this experiment, we use a generic design document on two different data sources to generate two different LDPs. A general overview of the latter experiment is shown in Figure 6.3.

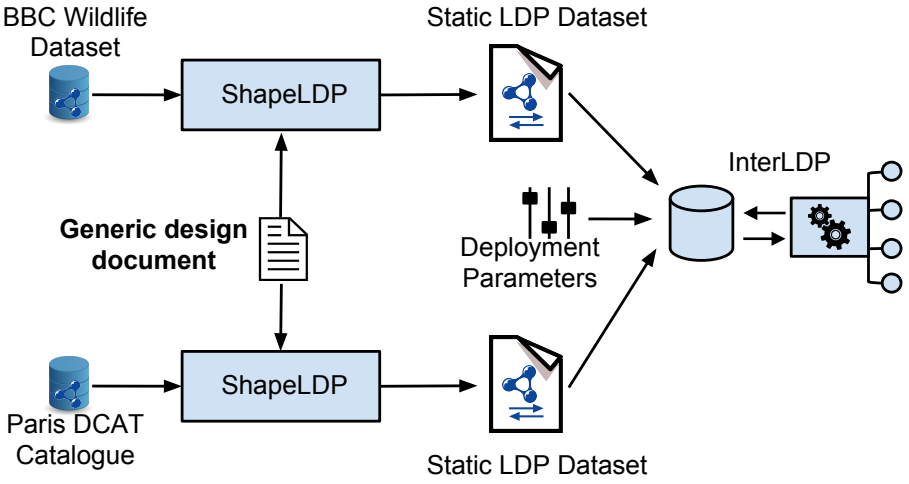

FIGURE 6.3: Overview of *Generic Design Reusability* experiment

Both data sources that we consider are in RDF. The first one is the *Wildlife Dataset*[1] from the BBC that provides details about biological species in RDF. The second one is the *Paris DCAT Catalogue*[2] and is used by Paris data portal to structure their data catalogue.

The design document used in this experiment is shown in Listing 6.2. It is a generic design as it can be applied on any RDF graph that uses the RDFS/OWL vocabulary which is general purpose. In brief, it contains a `ContainerMap :classCM` that defines containers for describing classes from the data source. `:classCM` in turn contains one `NonContainerMaps :instanceNM` and one `ContainerMap :subclassCM`. `:instanceNM` defines non-containers that describe class instances while `:subclassCM` defined containers that describe classes' subclasses. Subclasses may recursively have other subclasses. Consequently, `:subclassCM` has a `ContainerMap` that refer to itself (Listing 6.2, Line 34) so as recursively define LDP resources for describing the subclasses of all subclasses. Moreover, as we can see, all query patterns for extracting related resources uses only RDF and RDFS terms.

```
1 @prefix ldl: <https://w3id.org/ldpdl/#> .
2 @prefix : <http://example.com/data/> .
3 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
4 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
5
6 # :resourcesCM describes a single container for grouping all containers
7 # generated from :classCM
8 :resourcesCM a ldl:NullContainerMap;
9   ldl:slugTemplate "resources";
10  ldl:containerMap :classCM .
11
12 # :classCM describes containers for classes
```

[1] https://github.com/rdmpage/bbc-wildlife, last accessed on 23 June 2018
[2] https://opendata.paris.fr/api/v2/catalog/exports/ttl

```
:classCM a ldl:ContainerMap;
  ldl:resourceMap :classRM;
  ldl:nonContainerMap :subclassCM;
  ldl:nonContainerMap :instanceCM .

:classRM a ldl:ResourceMap;
  ldl:resourceQuery "{ ?x rdf:type ?{res} .}";
  ldl:graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o . }" .

# :classCM describes non-containers for instances of classes
:instanceCM a ldl:NonContainerMap;
  ldl:resourceMap :instanceRM .

:instanceRM a ldl:ResourceMap;
  ldl:resourceQuery "{ ?{res} rdf:type ?{parent} . }";
  ldl:graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o . }" .

# :subclassCM describes recursively containers for every suclass of a
    class
:subclassCM a ldl:ContainerMap;
  ldl:resourceMap :subClassRM;
  ldl:nonContainerMap :instancesCM;
  ldl:containerMap :subclassCM .

:subClassRM a ldl:ResourceMap;
  ldl:resourceQuery "{ ?{res} rdfs:subClassOf ?{parent} .}";
  ldl:graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o . }" .
```

Listing 6.2: Generic Design for RDFS/OWL Dataset

The design document is evaluated with respect to to the data sources using ShapeLDP to generate static LDP datasets from which LDPs are instantiated using InterLDP. The first LDP[1] expose resources from the BBC wildlife dataset while the second expose resources from Paris DCAT catalogue[2].

In this experiment, we have shown LDP-DL design reusability by using the same design document on two data sources that are structured using completely two different ontologies. This was possible because the design document was generic in the sense that it was based on concepts from RDFS and OWL that are standard ontology languages for RDF data.

Modular Design Reusability

In this experiment, we generate an LDP from a design document that reuses several modular design documents that may be used independently of each other. A general overview of the latter experiment is shown in Figure 6.4.

We consider a data source from the LodPaddle[3] project that provides RDF data generated from open data related to the city of Nantes in France. The data source consists of data about the different aspects of the city such as parkings, restaurants, cinemas. To define an LDP for that data source, we modularize the LDP-DL design

[1] http://opensensingcity.emse.fr/ldpdfend/d6/bbc/life/classes, last accessed on 29 June 2018

[2] http://opensensingcity.emse.fr/ldpdfend/paris/d6/classes, last accessed on 29 June 2018

[3] http://lodpaddle.univ-nantes.fr/lodpaddle/, last accessed 26 June 2018

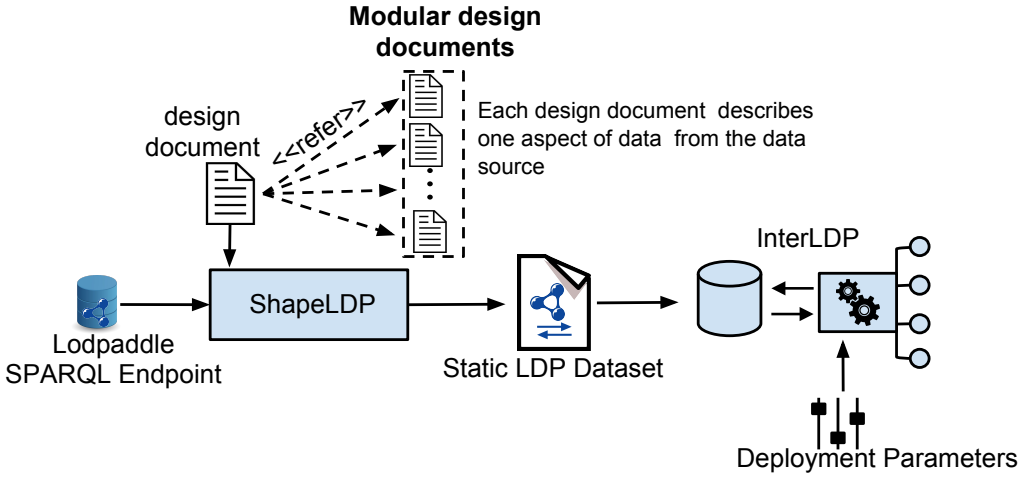

FIGURE 6.4: Overview of *Modular Design Reusability* experiment

into several design document each corresponding to a particular aspect of the data. Then, we have a main design document that refer to each modular design document. Part of the main design document is shown in Listing 6.3. The full main design document and the different modular parts are provided in Appendix D.2.

```
@prefix ldl: <https://w3id.org/ldpdl/#> .
@prefix :<http://example.com/data/> .

# definition of design document and links to its top-level maps
<> a ldl:DesignDocument;
  ldl:topLevelMap :transportCM,
          :cultureCM,
          :pointOfInterestCM .

# :transportCM describes a single container that groups containers related to the transport theme
# :mobilitesCM and :parkingsCM are described in a different document
:transportCM a ldl:NullContainerMap;
  ldl:slugTemplate "Transport";
  ldl:containerMap :mobilitesCM;
  ldl:containerMap :parkingsCM .

# :cultureCM describes a single container that groups containers related to the culture theme
# :cinemasCM and :bibliotheque_mediathequesCM are described in a different document
:cultureCM a ldl:NullContainerMap;
  ldl:slugTemplate "Culture";
  ldl:containerMap :cinemasCM;
  ldl:containerMap :bibliotheque_mediathequesCM .

# :pointOfInterestCM describes a single container that groups containers related to point of interests
# :chateau_monumentsCM and :parc_animalier_themesCM are described in a different document
:pointOfInterestCM a ldl:NullContainerMap;
  ldl:slugTemplate "PointOfInterest";
  ldl:containerMap :chateau_monumentsCM;
  ldl:containerMap :parc_animalier_themesCM .
```

LISTING 6.3: Main document containing partial LDP-DL design that links to external *maps

We organize the data using the organization that Lodpaddle uses in its application. In other words, we group data about some aspects of the city of Nantes in a particular container. For example, as shown in Listing 6.3, `data:transport` defines a container that groups all data related to transport. It then contains two other `ContainerMaps` `data:parkingsCM` (Listing 6.5) and `data:mobilitesCM` (Listing 6.4), that define LDP resources related to parkings and mobility respectively. The latter `ContainerMaps` are modularized in two different design documents that are shown in Listing 6.5 and Listing 6.4 respectively. Likewise, `data:cultureCM` and `data:pointOfInterestCM` groups data related to culture and point of interest in the city of Nantes and refer to other modularized design documents.

```
@prefix ldl: <https://w3id.org/ldpdl/#> .
@prefix : <http://opensensingcity.emse.fr/LDPDesign/data/> .
@prefix s:<http://lodpaddle.univ-nantes.fr/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

# :mobilitesCM describes a single container for grouping non-containers
    described by :mobiliteCM
:mobilitesCM a ldl:NullContainerMap;
   ldl:nonContainerMap :mobiliteCM .

:mobiliteCM a ldl:NonContainerMap;
  ldl:resourceMap :mobiliteRM .

:mobiliteRM a ldl:ResourceMap;
 ldl:resourceQuery "{ ?{res} rdf:type s:mobilite .}";
 ldl:graphQuery "CONSTRUCT { ?{res} ?p ?o . ?s ?p ?{res} . } WHERE { { ?{
     res} ?p ?o . } UNION { ?s ?p ?{res} . } }" .
```

Listing 6.4: Document containing partial design describing `:mobiliteCM`

```
@prefix ldl: <https://w3id.org/ldpdl/#> .
@prefix :<http://opensensingcity.emse.fr/LDPDesign/data/> .
@prefix s:<http://lodpaddle.univ-nantes.fr/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

# :parkingsCM describes a single container for grouping non-containers
    described by :parkingCM
data:parkingsCM a :NullContainerMap;
   :slugTemplate "parkings";
   :nonContainerMap data:parkingCM .

data:parkingCM a :NonContainerMap;
  :resourceMap data:parkingRM .

data:parkingRM a :ResourceMap;
 :resourceQuery "{ ?{res} rdf:type s:parking .}";
 :graphQuery "CONSTRUCT { ?{res} ?p ?o . ?s ?p ?{res} . } WHERE { { ?{res
     } ?p ?o . } UNION { ?s ?p ?{res} . } }" .
```

Listing 6.5: Document containing partial design describing `:parkingsCM`

We use ShapeLDP and provide it with all the main design document as well as all the modularized design documents in one go. As mentioned before, in such a case, ShapeLDP consumes all the design document and combines it into a single

model use which it generates the static LDP dataset that is finally deployed as an LDP[1] using InterLDP.

Discussion

LDP designs in LDP-DL are reusable both at the *document level* and *construct level*. By document level, we mean that it is possible to reuse the same design document on different data sources while construct level means that it possible to reuse the definition of LDP-DL constructs (`ContainerMaps`, `ResourceMaps`, etc.) in different design documents.

Domain Design Reusability (Section 6.2.1) and *Generic Design Reusability* (Section 6.2.1) shows the reusability of design document at document level. In *Domain Design Reusability*, we show the reusability of design documents by using the same design document on different data sources. Moreover, in *Generic Design Reusability* (Section 6.2.1), we provide a generic design that is completely agnostic of the underlying domain vocabulary and which is reusable on RDFS/OWL vocabularies. Compared to *Domain Design Reusability*, *Generic Design Reusability* shows the reusability of design documents at a more higher level.

However, there is a limitation with the generic design document used in *Generic Design Reusability*. If an RDF graph has a cycle in class-subclass hierarchy, using the generic design document, formally, an infinite LDP dataset will be generated. In our implementation, ShapeLDP proceeds to generate an infinite LDP dataset in the latter case. Normally, this would lead to a memory dump. This could be prevented by for example, setting a maximum depth on the recursive definition in the `data:subclassCM`. Theoretically, this would prune the inifite LDP dataset and return only a subset of it. However, for now, we do not tackle this issue in our implementation.

Modular Design Reusability (Section 6.2.1) shows the reusability at the construct level. In the latter experiment, *maps are defined in different design documents and are then referenced in the main design document without explicitly redefining them using their IRI. For now, ShapeLDP requires all documents containing the referenced instances of the LDP-DL constructs supplied to it during the evaluation.

It is possible to build the design model directly from the main design document without having to supply the URLs of the modular document. This may be done by publishing resources form the modular documents via LDPs. Then, when processing the main design document, an LDPizer may dereference instances of LDP-DL constructs from these LDPs and build the complete design document before evaluating it. However, for now, ShapeLDP do not implement the latter optimization which is why it explicitly requires the URLs of modular design documents.

6.2.2 Hosting Constraints

As mentioned before, there are data sources whose exploitation may give rise to hosting constraints preventing users from deploying the data from these sources in a different software environment. In this section, in Section 6.2.2, we generate an

[1] `http://opensensingcity.emse.fr/ldpdfend/nantes/custom/data/`, last accessed on 30 June 2018

LDP from a real-time data source to show the ability of our approach to deal with hosting constraints. We choose a real-time data source as it is a common example where such constraints may occur due to resources limitation (e.g bandwidth) to host and maintain fresh copies of the data. Finally, in Section 6.2.2, we evaluate our approach with respect to hosting constraints.

Dynamic LDP

The aim of this experiment is to show the ability of our approach to exploit a data source from its original location without actually hosting it in a different software environment. A general overview of this experiment is shown in Figure 6.5.

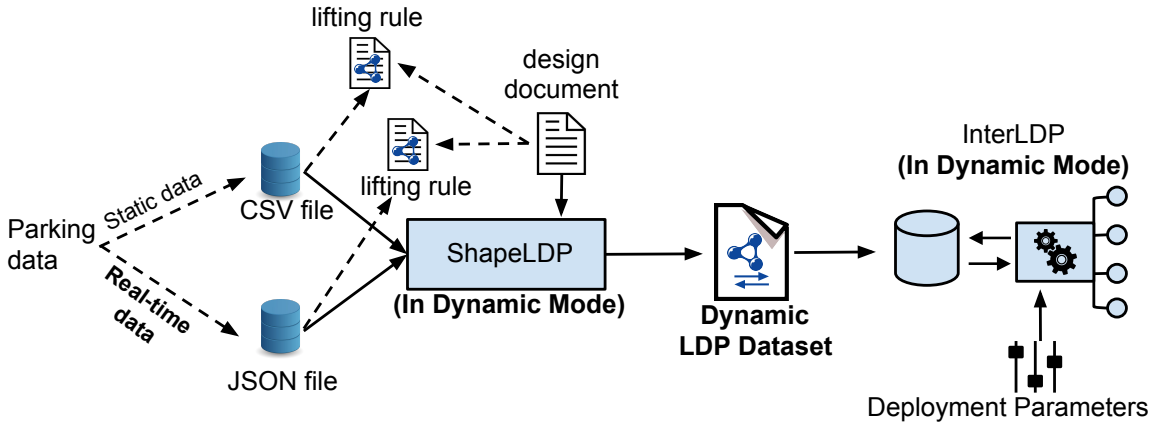

FIGURE 6.5: Overview of *Dynamic LDP* experiment

We consider a data source from Grenoble data portal[1] that provides details about availability of parking spaces in park and ride facilities. The data source itself consists of two parts, a static and a dynamic part. The static part[2] (in CSV) describes the different parking facilities together with their geographic coordinates. The dynamic part[3](in JSON) provides the number of free parking spaces in the parking facilities.

In this experiment, our constraint is that we do not want to host data from the real-time data source in our environment due to problems such as synchronization or updates. Therefore, using a design document (Appendix D.3) that specifies the lifting rules for the two data sources, we operate ShapeLDP in dynamic mode and generate a dynamic LDP dataset. Finally, we instantiate an LDP[4] from the dynamic LDP dataset using InterLDP in dynamic mode. The content of every resource exposed from the LDP is generated using data fetched from the data source at query time.

[1] http://data.metropolegrenoble.fr/ckan/dataset/parkings-relais-p-r, last accessed 30 June 2018

[2] http://data.metropolegrenoble.fr/ckan/dataset/parkings-relais-p-r/resource/92d894c5-f076-4973-9878-9215e8628e37, last accessed 30 June 2018

[3] http://data.metropolegrenoble.fr/ckan/dataset/parkings-relais-p-r/resource/457aa891-b53e-4a5d-aa9b-1be62201c088, last accessed 30 June 2018

[4] http://opensensingcity.emse.fr/ldpdfend/grenoble/parkings

Discussion

To deal with hosting constraints, we use the dynamic LDP datasets. In this way, we avoid storing the content of the constrained data source in a different environment and at query time, we dynamically exploit the latter data source.

The data source for which an dynamic LDP dataset is generated may not necessary be a real-time one, it may be also static. Both the language and our implementation are completely agnostic of the update frequency of the source which allow us to consider any type of constraints (license, storage, update frequency, etc.).

However, as mentioned in Section 4.3.2, dynamic LDP dataset can only handle graph pattern changes. Thus, if a data source have hosting constraints and with respect to a design document, there are query pattern changes, dynamic LDP dataset may not be the solution. Using our approach, the only way to deal with graph pattern changes for now is to re-generate the entire LDP dataset for every request.

6.2.3 Data Heterogeneity

In this section, our aim is to show that our approach can generate LDPs from heterogeneous data sources while being able to use the data from these sources either altogether or separately in content of LDP resources. While the experiment in Section 6.2.2 was focused on hosting constrains, it also showed the generation of an LDP from two heterogeneous data sources where the content of all resources contained data from both sources. In complement to that experiment, in this section, we perform an experiment (Section 6.2.3) focused on date heterogeneity in which an LDP is generated from two heterogeneous data sources and the content of all LDP resources contain data only from one of the source. Finally, in Section 6.2.3, we discuss the results both experiments in to evaluate the ability of our approach to deal with heterogeneous data sources.

Heterogeneous LDP Generation

The aim of this experiment is to deploy an LDP from two data sources that are in two different formats namely CSV and JSON. A general overview of this experiment is shown in Figure 6.6. The CSV data source[1] provides details about service stations for shared vehicles while the JSON data source[2] provides station for bicycle sharing.

In the design document, we specify RDF lifting rules of the CSV and JSON data source in SPARQL-Generate [LZB17b]. We provide the design document and lifting rules in Appendix D.4. ShapeLDP evaluates the design document with respect to the data sources and use the embedded SPARQL-Generate engine to generate the RDF data that it uses to generate the static LDP dataset. Finally, InterLDP instantiate an LDP from the static LDP dataset and some deployment parameters.

In the LDPs generated, resources are exclusively generated from either the CSV or JSON data source.

[1] https://opendata.paris.fr/explore/dataset/liste-des-stations-de-services-de-vehicules/, last accessed 30 June 2018

[2] https://opendata.paris.fr/explore/dataset/velib-emplacement-des-stations, last accessed 30 June 2018

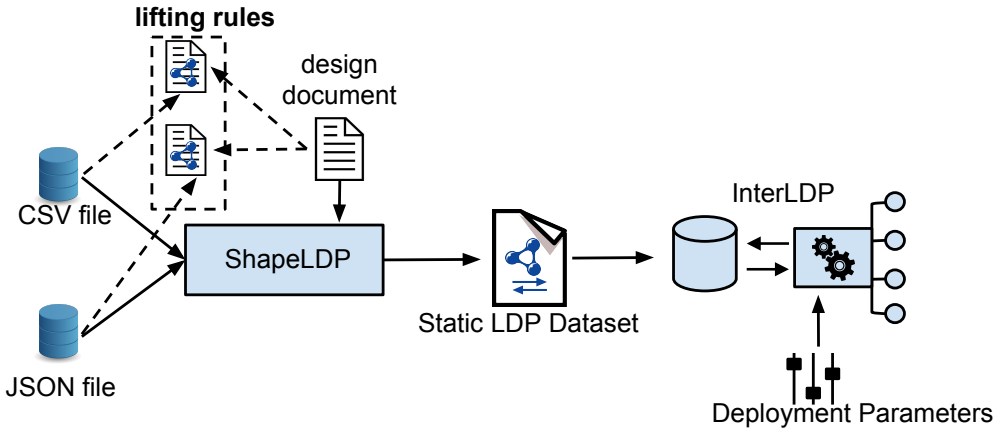

FIGURE 6.6: Overview of *Heterogeneous LDP Generation* (Section 6.2.3)

Discussion

We have done two experiments in which we have generated LDPs from heterogeneous data sources. In the first experiment (Section 6.2.2), though focused on hosting constraints, an LDP was generated where all its resources use data from all sources. In the second experiment (Section 6.2.3), the content of LDP resources originated from only one source. With these two experiments, we showed that we are able to exploit heterogeneous data sources with the complete freedom to use their content in any way when generating RDF resources. This is possible because when data from heterogeneous data sources are converted and merged into a single RDF graph, the data can be used independently of its source due to the flexibility of the RDF data model.

Also, in the latter experiments, the RDF lifting rules were expressed in SPARQL-Generate because, as mentioned in Chapter 5 (Section 5.3.1), for now, ShapeLDP can only consume lifting rules in this language. However, this does not limit our ability to deal with data heterogeneity because as mentioned in Chapter 2 (Section 2.1.2), SPARQL-Generate can be extended to any heterogeneous data source.

Moreover, the model of LDP-DL is itself open to heterogeneous data sources irrespective of any implementation. Per the syntax of LDP-DL, a `ResourceMap` can exploit any data source, whether in RDF or not, as long as RDF lifting rule is specified for a data source.

6.2.4 Automated LDP Generation

As mentioned before, LDP generation include both the design and deployment phase. However, in previous experiments, both processed in both the design and deployment phase were automated using our tools (ShapeLDP and InterLDP). In the design phase, we use ShapeLDP only as it is the only LDPizer that exist so far. In the deployment phase, so far we have deploy LDPs only using InterLDP. However, besides InterLDP, there are other existing LDP servers on which the deployment of LDPs can be automatized.

To this end, in this section, we describe an experiment (Section 6.2.4) in complement to the previous ones where we automate the deployment of an LDP on Apache

Marmotta which is a referenced LDP implementation. Since Apache Marmotta also allow write operations, we take the opportunity to show that our approach also allow updating LDPs. Finally, in Section 6.2.4, we evaluate the ability of our approach to automate the generation of LDPs using the results of all experiments done so far.

Compatible LDP Generation

In this experiment, we use ShapeLDP to generate a dynamic LDP dataset from a design document with respect to a data source. Then, we use POSTerLDP to deploy the dynamic LDP dataset on an LDP server that is an instance of Apache Marmotta. Then, using the same dynamic LDP dataset, we update the LDP using POSTerLDP. An overview of this experiment is shown in Figure 6.7.

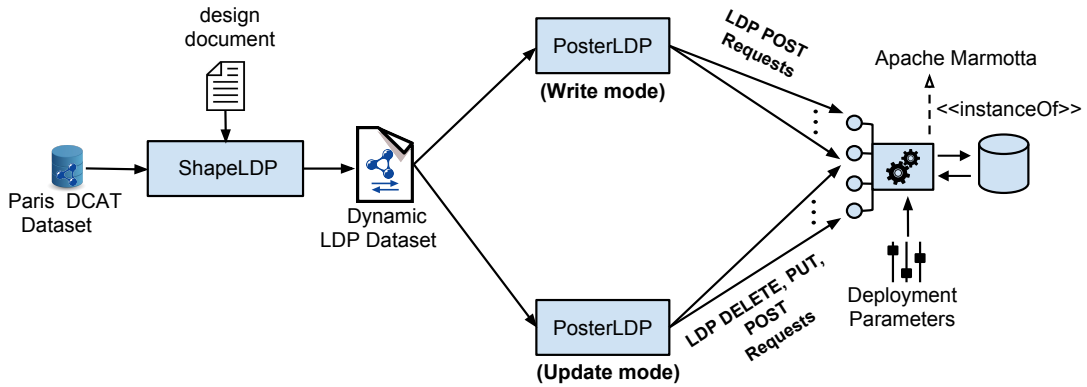

FIGURE 6.7: Overview of *Compatible LDP Generation*

We use the data source and the design document from *Domain Design Reusability* (Section 6.2.1).

When using POSTerLDP to deploy the dynamic LDP dataset, we use the `empty` option that, as described in Section 5.4.1, enable the creation of LDP resources with empty RDF graphs. Then, when using POSTerLDP to update the LDP, we use the same dynamic LDP dataset but without using the `empty` option that consequently updates the content of all resources on the server.

Using the same dynamic LDP dataset is possible both for writing and updating LDP resources because as mentioned in Chapter 5 (Section 4.3.2), in this structure, rather the RDF graphs, CONSTRUCT queries for generating the RDF these graphs are materialized. Consequently, when using the dynamic LDP dataset, POSTerLDP generates a new LDP dataset that may contain new RDF graphs for LDP resources if the data source has changed. In our case, since the data source is static, using a dynamic LDP dataset will not change the content of LDP resources which is why we use the `empty` option to clearly see the results of updating their content.

The aim of this experiment is to show the compatibility of our approach with existing LDP implementation. We have shown this aspect by deploying the an LDP using Apache Marmotta. Any LDP server could have been used instead of Apache Marmotta as POSTerLDP is completely agnostic of the server or its environment and can interact with server implementing LDP interactions.

Discussion

As mentioned before, the development of LDPs consists of two main phases namely the design and deployment phase. To automate the generation of LDPs, both these phases need to be automatized. In all our experiment described in Section 6.2.3, Section 6.2.2 and Section 6.2.1, we have shown the ability of our approach to automatize the deployment of LDPs from data sources that are in RDF, are heterogeneous or have hosting constraints. In all the latter experiments, the deployment was automatized using InterLDP. However, in *Compatible LDP Generation* (Section 6.2.4), POSTerLDP was used to automatized the deployment on the instance of Apache Marmotta both in write and update mode. As mentioned before, POSTerLDP is completely agnostic of the implementation of the server and can therefore be used to automate deployment on any LDP conformant server.

In our approach, we have not explicitly automatized the design phase as doing so requires automatizing the design decision making process that involves making decisions with respect to standards and best practices. However, for most of the design decisions that have to be taken when writing LDP design in LDP-DL such as resource organization in containers or content of containers or non-containers, there are no standards. To be able to justify automated design decisions, there is a need to identify design patterns that is beyond the scope of our work.

Nevertheless, as shown in *Generic Design Reusability* (Section 6.2.1), we indirectly automatize the design phase through the use a generic design containing design decisions taken based on our intuition and experience. For now, the generic design can be used on RDFS/OWL vocabularies. But we can think of having generic design any vocabularies such as SKOS [MB09] or even for cross-domain vocabularies such as Time ontology [CL17], PROV-O ontology [LSM12].

In summary, with respect to design phase, our approach can automatize it partially. The deployment phase on a single server is fully automatized both in write and update mode as long as the sever is LDP conformant.

6.3 Side Contributions

Besides satisfying our evaluation criteria as described in the previous section, our approach has several other side contributions that we discuss in this section. In Section 6.3.2, we discuss the ability of our approach to perform lightweight data integration. Then, in Section 6.3.1, we describe the flexibility of LDP-DL by defining several LDP designs for the same data source. Finally, we describe our LDP server InterLDP in Section 6.3.3 as an additional compatible LDP implementation. This section follows the same layout as the previous one with section title referring to name of experiments and figures providing overview of experiments.

6.3.1 Flexibility

The flexibility of our approach is its ability to adapt to changes. In this section, we perform an experiment (Section 6.3.1) to demonstrate the flexibility of LDP-DL to cope with the need to change an LDP design by showing that LDP-DL allows defining several designs for the same data source. Finally, in Section 6.3.1, we discuss

the flexibility of LDP-DL and our approach in general using results of experiments done so far.

Flexible Design

In this experiment, we use the 21 DCAT catalogues used in *Domain Design Reusability* (Section 6.2.1) as data sources. We define five LDP-DL designs based on the DCAT ontology and use each of them with respect to the data sources. For each of the data source, we generate five LDPs with different designs. A general overview of the latter experiment is shown in Figure 6.2.

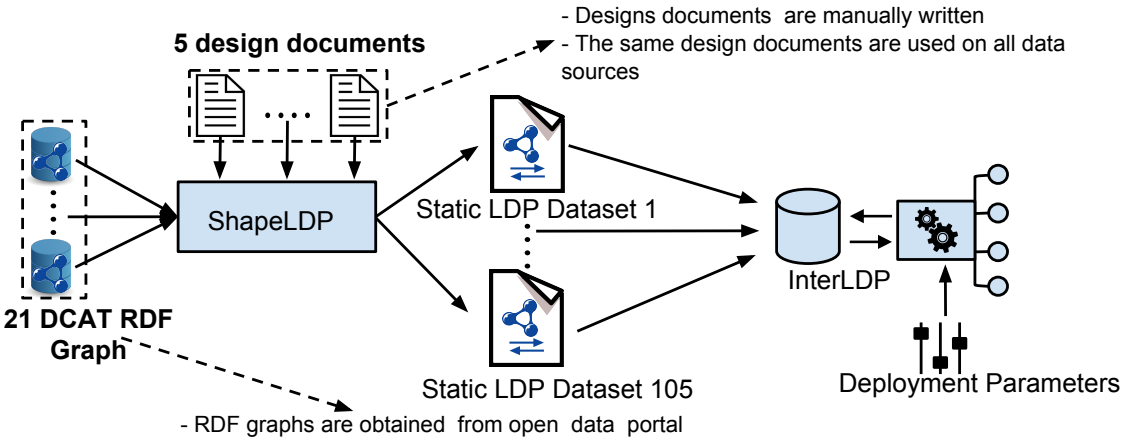

FIGURE 6.8: Overview of *Flexible Design* experiment

The input data sources are RDF graphs that are structured per the DCAT vocabulary [ME14b]. We obtain these RDF graphs by considering the list of open data portals curated by OpenDataSoft[1] that provide a description of their catalogues using the DCAT vocabulary. In all, we obtain 21 RDF graphs. Then, we consider five LDP designs and encode them using the concrete syntax of LDP-DL in five design documents. Each design describes a way to publish resources from an RDF graph structured per the DCAT vocabulary via an LDP.

The first design document is the same as the one used in *Domain Design Reusability* (Section 6.2.1). All remaining four designs builds on the latter design. The second design defines explicit containers to group containers that describes DCAT datasets and distributions. The third design is mostly similar to the second one except that in the containers that describe DCAT catalogs, it defines containers for grouping datasets based on their languages. The difference between the third and fourth design is that the latter defines containers for grouping datasets based on their themes rather than languages. Finally, the fifth design is a combination of the third and fourth design in that it defines containers for grouping DCAT datasets both based on their languages and themes. In Appendix D.5, we provide all the remaining four design documents and describe how to access all the LDPs (105 in all) generated from the five design documents.

[1] https://www.opendatasoft.com/a-comprehensive-list-of-all-open-data-portals-around-the-world/, last accessed on 5 January 2018

In this experiment, the aim was to show the flexibility of LDP-DL by defining different designs for the same data source. We achieved this by defining five different designs from the same data source and by generating different LDPs using them.

Discussion

In the previous experiment, we have showed that LDP-DL is flexible enough to allow different LDP designs to be defined for the same data source. This flexibility can be useful in different use cases. For example, currently several ontologies are being standardized such as Time ontology, PROV-O ontology, etc. Users may write different LDP designs to publish their data structured per a particular ontology. As a result, for that particular ontology, several LDP-DL designs may be written and out of them, those designs that follow the proper standards, design patterns and best practices may emerge.

Besides the flexibility of LDP-DL, there are properties of our models and tools that enhances the flexibility of our approach. If we consider the experiment in *Dynamic LDP* (Section 6.2.2) where we expose LDP resources from a real-time data source, a change in the data source is directly reflected in the content of the LDP resource due to the use of a dynamic LDP dataset exposed using InterLDP in dynamic mode. Moreover, as we saw in Compatible LDP Generation, our approach may still be able to adapt even if an LDP server other than InterLDP is used by using POSTerLDP to update the content of existing LDP resources. This is possible because POSTerLDP is agnostic of the implementation of any servers and interact with them using LDP interactions. In summary, the flexibility of our approach enables adapting to changes in the LDP design, data source and LDP server.

6.3.2 Lightweight Data Integration

In Chapter 1 (Section 1.3.2), we explained difficulties that may be raised in highly decentralized information ecosystem during data integration. In this section, in Section 6.3.2, we perform an experiment to show the ability of our approach to facilitate data integration. Finally, in Section 6.3.2, we discuss the use of our approach to perform lightweight data integration in general.

LDP Integration

In this experiment, we use a design document whose data source describes the 105 LDPs generated in *Flexible Design* (Section 6.3.1) to generate another LDP that integrates all of them. Figure 6.9 shows a general overview of this experiment.

As mentioned in Chapter 5 (Section 5.3.2), LDP datasets contain metadata in their default graph which is also exposed by a specific RDF source on the LDP. In fact, the data source contains the union of this metadata for all the LDPs generated in *Flexible Design* (Section 6.3.1). Thus, the data source[1] provide information about the top level resources, containers and non-containers of all LDPs.

The design document is shown in Listing 6.6. The `ContainerMap` `:LDPInstanceCM` defines containers for every LDPs. It has a `NonContainerMap` `:LDPTopResNM` that

[1] `https://github.com/noorbakerally/LDPDatasetExamples/blob/master/LDPsInfo.rdf`, last accessed 7 October 2018

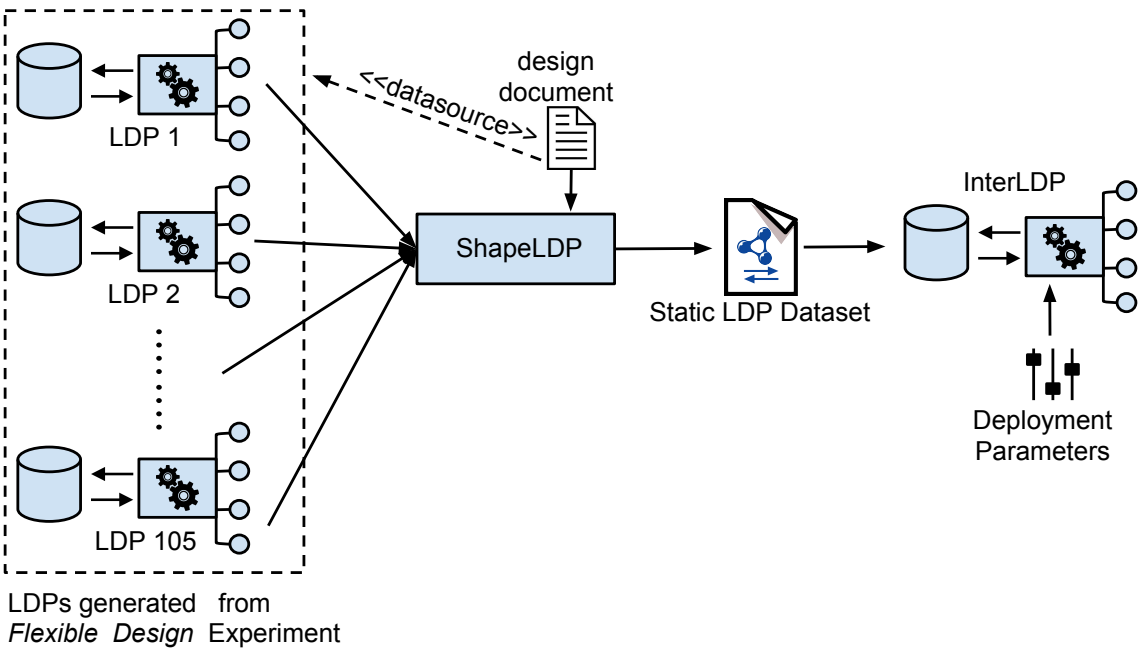

FIGURE 6.9: Overview of *LDP Integration*

defines an LDP RDF source for every top level resource for a corresponding LDP. Notice the graph query of the `ResourceMap :LDPTopResRM` (Line 21). `ldl:refer` is a special predicate we use to indicate that the resource in the object position is an external LDP resource. `ldp:contains` cannot be used because per the LDP standard it can only be used when the LDP resource is hosted within the current LDP. The LDP browser HubbleLDP uses this predicate to recognize these external LDP resources.

```
1 @prefix ldl: <https://w3id.org/ldpdl/ns#> .
2 @prefix : <http://example.com/data/>
3
4 # :LDPInstanceCM describes a container for every LDP dataset
5 :LDPInstanceCM a ldl:ContainerMap;
6    ldl:resourceMap :LDPInstanceRM;
7    ldl:nonContainerMap :LDPTopResNM .
8
9 :LDPInstanceRM a ldl:ResourceMap;
10   ldl:dataSource :LDPsInfoDS;
11   ldl:resourceQuery "{ ?{res} a ldl:LDPDataset . }";
12   ldl:graphQuery "CONSTRUCT { ?{res} ?p ?o . ?s ?o ?{res} . } WHERE {?{
         res} ?p ?o . ?s ?o ?{res} .}" .
13
14 # :LDPTopResNM describes a container for every top-level resource in an
     LDP dataset
15 :LDPTopResNM a ldl:NonContainerMap;
16   ldl:resourceMap :LDPTopResRM .
17
18 :LDPTopResRM a ldl:ResourceMap;
19   ldl:dataSource :LDPsInfoDS;
20   ldl:resourceQuery "{ ?{parent} ldl:topLevelResource ?{res} . }";
21   ldl:graphQuery "CONSTRUCT { ?{nres} ldl:refer ?{res} } WHERE {}" .
22
23
```

```
24 :LDPsInfoDS a ldl:RDFFileSource;ldl:location "https://raw.
       githubusercontent.com/noorbakerally/LDPDatasetExamples/master/
       LDPsInfo.rdf" .
```

Listing 6.6: Design document for lightweight integration

The aim of this experiment was to show that LDPs deployed on different servers can be all integrated in a single LDP. This is achieved as in the resulting LDP, at the top level, there are containers for every LDPs that in turn contains a reference to their top level resources. All the remaining containers and non-containers on the 105 LDPs can be accessed using these references.

Discussion

In Chapter 1 (Section 1.3.2), we explained the effort and resource duplication raised when integrating data from heterogeneous data sources. As shown in this experiment, it is possible to overcome these problems using LDPs whose generation is facilitated by providing direct access to the required metadata.

In the latter experiment, we have shown that using an LDP dataset of some kilobytes, we are able to generate an LDP that integrates resources from hundreds of LDPs. Navigating through the platform gives the impression as if all its resources are hosted on a single server which is not the case.

The linked data platform can integrate LDP resources from the LDPs without having to bother about the amount of hardware resources required to store or process them. Any request for an LDP resource hosted on a different LDP server automatically goes to that server instead of the linked data platform. Thus, the server hosting the linked data platform can itself be a lightweight server that requires the bare minimum resources to instantiate InterLDP and process request only for resources that it hosts. In this way, the servers share hardware and software resources among themselves and integrate data from each other.

The lightweight data integration is in fact a feature that is possible only when data is published following Linked Data principles. Following the LDP standard facilitate the process by making different autonomous servers understand and communicate with each other and may thus be beneficial to both decentralized and highly decentralized information ecosystem. In this regard, our approach facilitates the putting in place of such servers, thus an information ecosystem where lightweight data integration is possible.

6.3.3 InterLDP as an LDP Implementation

InterLDP is a fully compatible LDP server and therefore it adds to the existing implementations of LDP. From the LDP domain model, it implements RDF sources and basic containers. Also, from the LDP interaction model, it implements only read operations on RDF sources and basic containers. To validate LDP implementations for the LDP standard, the W3C provides a test suite[1] with a number of test cases for verifying whether LDP interaction model are properly implemented on the different types of resources from the LDP domain model. InterLDP validate part of the

[1] https://w3c.github.io/ldp-testsuite/ accessed on 25 November 2017

test cases corresponding to the LDP interactions it implements as detailed in its execution report[1] generated by the test suite.

6.4 Summary

In this chapter, we have described the evaluation of LDP-DL, its implementation and the approach of automatizing the generation of LDPs in general. In our evaluation, we have described several experiments carried out with the aim of demonstrating one or more features of our language, implementation and/or approach. We have discussed the observations and results made in these experiments with respect to to the evaluation criteria and we have detailed side contributions made as well. Table 6.2 summarizes these experiments with respect to our evaluation criteria.

	Evaluation Criteria			
	DH	**HC**	**DR**	**A**
Domain Design Reusability			✓	✓
Generic Design Reusability			✓	✓
Modular Design Reusability			✓	✓
Heterogeneous LDP Generation	✓			✓
Dynamic LDP	✓	✓		✓
LDP Integration				✓
Flexible Design				✓

DH:Data Heterogeneity **HC**:Hosting Constraints
DR:Design Reusability **A**:Automatization

Table 6.1: Summary of experiments

	Evaluation Criteria			
	DH	**HC**	**DR**	**A**
Homogeneous LDP Access	✓			✓
Dynamic LDP	✓	✓		✓
Domain Design Reusability			✓	✓
Generic Design Reusability			✓	✓
Modular Design Reusability			✓	✓

DH:Data Heterogeneity **HC**:Hosting Constraints
DR:Design Reusability **A**:Automatization

Table 6.2: Summary of experiments

With respect to *Design Reusability*, we made three experiments that validate our approach with respect to design reusability. The first two and the last one show design reusability at the document and construct level respectively. In the first one, we showed the reusability of an LDP-DL design on different data sources structured per the DCAT vocabulary. The second one shows the reusability of the same generic design on different data sources that uses RDFS/OWL ontologies. The third one

[1] https://w3id.org/ldpdl/InterLDP/execution-report.html

shows the reusability of LDP-DL constructs from several LDP-DL designs to build a different LDP-DL design.

To satisfy *Data Heterogeneity*, we generated an LDP from heterogeneous data sources. For now, the ability of our approach to consider heterogeneous data sources entirely depends on the use of RDF lifting rules that provide an RDF view of these sources. With respect to *Hosting Constraints*, we validate this criteria by instantiating LDPs from dynamic LDP datasets for exploiting data sources having hosting constraints at runtime without having to host them. As we mentioned before, there are changes that may occur at these data sources that dynamic LDP datasets cannot handle. For now, our approach can only naively handle them by re-generating the dynamic LDP datasets every time an LDP request is made. Also, as we can see, all experiments satisfy the *Automatization* criteria as the LDPs generated in them were automatized.

Besides our evaluation criteria, our approach provides some side contributions. In *LDP Integration*, we show the ability of our approach to generate LDPs that integrates resources from several other LDPs by directly referring to them. Though not shown in our experiment, this gives the possibility of performing cascading LDP integrations. Also, with respect to *Flexible Design*, our language can express different designs for a particular data source. Finally, InterLDP that we have used in different experiments in an LDP server that adds to existing LDP implementations.

Part IV

Conclusion & Perspectives

Chapter 7

Conclusion

Contents

7.1 Summary of Contributions

The use of information systems in organizations started with centralized architectures. The problems with such architectures were quickly realized by its users as they were difficult to maintain and often ended as big systems with complex structures. Decentralized information ecosystems emerged as a solution with smaller information systems delineated towards specific areas of responsibilities. The reason for their success was that the information systems could exchange data as data providers were collaborating with data consumers to help them making sense of the data whenever required. Today, information ecosystems in which organizations are operating are moving towards highly decentralized architectures for various reasons such as globalization, multinationalism or complex supply chain collaborations. In order to study and characterize information ecosystems with these architectures, we introduced the new expression "highly decentralized information ecosystem".

Highly Decentralized Information Ecosystem We defined such an information ecosystem as a decentralized information ecosystem consisting of information systems managed by units that are self-governed, independent and exist with little to no coordination between them. The self-governance of individual units and lack of coordination create several problems. Firstly, the information systems may become isolated with no interaction between them. Consequently, data providers and consumers may be unaware of each other. In cases where data consumers are aware of data sources exposed by data providers, they may lack the required metadata to exploit them and handle their heterogeneity. Finally, unlike decentralized information ecosystem, data providers may not participate with data consumers to facilitate exploitation of their data sources.

Requirements for Interoperability In highly decentralized information ecosystem, enhancing interoperability is the core challenging issue. To tackle it, rather than providing tools for facilitating data consumption, we decided to abstract the data heterogeneity with a standard layer. Doing so both facilitates data consumption and provides a homogeneous platform upon which generic tools can be developed for different purposes such as discovery of data sources or data integration. Therefore, we identified the core requirements that this standard layer should satisfy to enhance data interoperability at syntactic, semantic and data access level. In short, at the syntactic level, there is a need for a standardized and flexible data syntax that enforces the use of IRIs for naming resources. At the semantic level, globally-scoped identifiers should be used to name resources and data semantics should be made explicit using ontology languages. Also, these ontology languages should allow encoding these semantics in the data itself to make it self-described and portable. Finally, at the data access level, there is a need for a high-level standardized data access together with the description to access it.

Semantic Web for Highly Decentralized Information Ecosystem Using the latter requirements as a basis, we showed that standards from the Semantic Web actually satisfy them and can therefore be used to implement this standard layer. We positioned the LDP standard as the final step to publish data at this layer.

We surveyed LDP related works and found out that they are limited and basic and requires much manual work to set LDPs. Therefore, we identified a set of requirements that some tools or approach should satisfy to automate the generation of LDPs from existing data sources that may be heterogeneous or have hosting constraints while ensuring that the design of these LDPs are reusable.

LDP Generation Workflow With these requirements in mind, we presented a generalized LDP generation life cycle based on model-driven engineering principles to facilitate the design and deployment of LDPs. Based on some core LDP design and deployment aspects and the generalized LDP generation life cycle, we proposed the LDP generation workflow that satisfies the requirements we identified for automatizing the generation of LDPs from existing data sources. The workflow automatizes both the design and deployment phase of LDPs and can be distributed as geographically dispersed actors may implement different parts of it. Core of the LDP generation workflow is a domain-specific language, LDP-DL, to describe the design of LDPs separate and independent of any implementation.

LDP Design Language We defined the syntax and the semantics of LDP-DL. The semantics was defined in a model-theoretic fashion with the aim of abstracting from choices left open by the LDP standard and by doing so, we defined the validity of an LDP-DL design interpretation. Then, we presented an example of a family of algorithms that implements a valid LDP-DL interpretation and characterizes the open choices. We also demonstrates the correctness of these algorithms using the validity of the LDP-DL design interpretation that we have defined. We also defined the LDP dataset model that is in fact the semantics of our language. It can be seen as a snapshot of an LDP at given time and while being a simple model, it enables abstracting from all implementation-specific details of LDPs. An LDP dataset can vary depending on the location of the server where it is deployed as well as the dynamicity of the data sources using which it was generated. To abstract this variability, we provide the static and dynamic LDP datasets. Static LDP dataset allows instantiating LDP datasets at deployment time by setting the location of servers while dynamic LDP datasets ensure that LDP datasets remain valid with respect to some changes in data sources used to generate them.

LDP Generation Toolkit We provided the LDP generation toolkit that is an implementation of the tools from the workflow and allows us to automate the generation of LDPs from existing data sources while remaining agnostic of the technicalities of LDP servers. The existing data sources include RDF and heterogeneous sources as well as those having hosting constraints. The ability to deal with data heterogeneity is a feature intrinsic to LDP-DL that is inherited by the implementation. Hosting constraints are dealt by using generating dynamic LDPs that interpret dynamic LDP datasets to generate LDP resources at query time. While designs in LDP-DL are mostly reusable by excluding the data source, the implementation further enhances its reusability by enabling the modularization of the design. Besides, flexibility and lightweight data integration are side contributions of the approach.

7.2 Limitations

Below, we describe some limitations of our approach.

Partial Coverage of LDP standard With respect to the LDP standard, we have considered only RDF sources and basic containers and are able to design and deploy LDPs. However, there are number of other important aspects to be addressed. While in one of our work [BZ17], we did consider Non-RDF sources, we did not integrate them in our model as we wanted to set up a good foundation for RDF sources first. Moreover, we did not consider direct and indirect containers that offer more flexibility than RDF sources and basic containers. Also, for now, our approach supports only read operations. Therefore, much work remains to be done to automatize the generation of read-write LDPs that covers all aspects from the standard. However, we believe that before these aspects can be considered, some ambiguities and confusions need to be resolved within the LDP standard itself. For example, regarding membership triples in direct containers and indirect containers, the LDP standard is not clear about the location to materialize them especially when membership resources are external resources. Moreover, while resources from other LDPs can be referred as RDF resources, there is no explicit way to refer to them as proper LDP resources that to some extent isolate LDPs from each other. Likewise, there are other aspects that need to be clarified and made explicit.

Generation of Design Documents is still manual While the design phase in the LDP generation workflow is automatized by LDPizers, design document still needs to be written. While generic LDP-DL designs can be used as shown in one of our experiments, the process of writing them is still manual and the designs decisions they encode are based on mere intuitions that may be erroneous. In short, we can say that the design phase can only be partially automatized using generic LDP-DL designs. Fully automatizing this process requires automatically taking decisions considering design patterns. While these patterns can be identified in existing Linked Data, doing so is a complex process and requires much resources. For example, much RDF data is available in RDF dumps whose size can be in the order of tenths to hundreds of gigabytes and processing them to look for patterns requires much resources in terms of processing power, disk storage and time given the size of Linked Data on the Web. We started this process but had to stop it midway due to time constraints. In short, to automatize the generation of design documents at least from RDF data, the identification of design patterns is a core step that remains to be done.

Complexity behind Lifting Rules In our approach, we abstract heterogeneous data sources using their lifting rules in RDF. By doing so, in some way, we make it appear that dealing with heterogeneous data sources is easy. Indeed, LDP-DL is open to any type of data sources and it defines lifting rules at a rather abstract level by remaining agnostic of the language in which they are written or systems that process them. However, before this lifting rule can be obtained, a number of tasks has to be undertaken and doing so may raise different problems. For example, to describe heterogeneous data, there is a need for ontologies that may be obtained

from repositories. However, they may have to be developed if they are not available and doing so can be a complex and lengthy process. Thus, generating LDPs may still be complex if the lifting rules for heterogeneous data are not available. In this work, while our focus on using LDP as a final step to publish data, there are still much efforts needed for facilitating the use RDF and RDFS/OWL at the data syntax and semantics level.

Invalidation of LDP datasets in Dynamic LDPs Currently, to deal with data having hosting constraints, we instantiate dynamic LDPs from dynamic LDP datasets and consequently, generate content of LDP resources from these sources at query time. However, in dynamic LDP datasets, the LDP resources that describe resources from these sources remain fixed. Consequently, if the sources change, there may be resources deleted at the sources while still being hosted by the LDP due to their existence in the dynamic LDP dataset. Similarly, there may be resources added at the sources but inexistent on the LDP as there are no resources describing them in the dynamic LDP dataset. We have described this problem in detail in Chapter 4 (Section 4.3.2) as we mentioned, for now, we only deal with changes at the sources that affect the content of LDP resources and not existence or inexistence of LDP resource themselves. From an abstract level, not handling such changes may invalidate the LDP dataset with respect to the design document. As we mentioned, for now, to ensure validity of LDP datasets, we can only regenerate them from dynamic LDP datasets every time an LDP request is obtained.

7.3 Perspectives

As we mentioned before, current scientific works on LDPs is very limited and to our knowledge, our work is the first attempt to automatize their generation. While having a number of limitations, it opens up avenues for future researches that we describe below.

Enriching design aspects in LDP-DL Model In this thesis, we considered only three design aspects namely the IRI, content and organization of LDP resources and have performed deployment of LDPs on one server. With these aspects, while we are able to design and deploy LDPs, our models can be extended with several other aspects from the LDP standard itself such as direct and indirect containers. Besides the LDP standard, there are other specifications such as LDP Paging 1.0 [SAM15b] or Linked Data Notifications [CG17] describing design or deployment aspects that could be used to further enhance our current model. Also, the LDP dataset model we introduced to abstract LDPs can be used as a basis for write operations. For example, we can define an algebra and theorize HTTP methods for performing both read and write operations as algebraic operators that use or manipulate the LDP dataset model. In short, while the models that we provide in this thesis are simple, they can server as a strong foundation for further extension depending on use cases. A security model could be a good start.

Linked Data Publication based on Best Practices Using our approach, existing data sources can be published via LDPs having different designs. However, for now, our approach provides no help in publishing Linked Data that follows best practices. Data on the Web Best Practices [LBC17] is a standard from the W3C that publishes best practices to enhance data interoperability between data publishers and consumers. These best practices relate to a number of aspects that data licensing, provenance, quality or versioning and for each of them, the standard describes possible approaches for implementing them. Therefore, it would be interesting to find a way to integrate this standard in our approach to enable the generation of Linked Data that implement these best practices.

LDP Generation Methodology As we mentioned before, design documents can be modularized and different actors may collaborate and work on their different modules in parallel. Consequently, there may be a lack of management when the workflow is instantiated in a highly decentralized information ecosystem with actors from different organization collaborating on a single design document to set up an LDP. Consider a situation where a government body responsible for a city wants to facilitate access to city data using an LDP. In this case, the government body may be the data publisher and the organizations in the city may be the data provider. The organizations may provide different parts of the design documents that may then be used to instantiate the LDP. From this point, there are different ways for setting up the LDP. While it is possible to wait for all organizations to provide their design modules and then proceed with the deployment, proceeding in an agile way where the LDP is set up as soon as an organization provide its design module may enable quicker deployment. In short, having an LDP generation methodology where the different sequential or parallel activities and the parties responsible for completing them have been identified may help in better managing projects.

Appendix A

Parking Example

In Chapter 2, we uses a parking ontology and provides it in Turtle. Here, we give its serialization in XML and JSON-LD below.

A.1 Parking Example XML Listing

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:dc="http://purl.org/dc/terms/"
       xmlns:ns0="http://www.opengis.net/ont/geosparql#"
       xmlns:ns1="http://example.org/ontology/"
       xmlns:gr="http://purl.org/goodrelations/v1#"
       xmlns:schema="http://schema.org/">

 <rdf:Description rdf:about="http://example.org/data/Anvers4">
   <rdf:type rdf:resource="http://example.org/ontology/ParkingFacility"/>
   <dc:title>Anvers</dc:title>
   <ns0:lat rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal
       ">48.83523833495664</ns0:lat>
   <ns0:long rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal
       ">2.37322158810141</ns0:long>
   <ns1:carParkingSpaces rdf:datatype="http://www.w3.org/2001/XMLSchema#
       integer">100</ns1:carParkingSpaces>
   <ns1:motocycleParkingSpaces rdf:datatype="http://www.w3.org/2001/
       XMLSchema#integer">50</ns1:motocycleParkingSpaces>
   <ns1:parkingManager>
     <ns1:ParkingManager rdf:about="http://example.org/data/Anvers">
       <gr:legalName>Anvers Parking</gr:legalName>
       <schema:telephone>0754234448</schema:telephone>
       <schema:email>parking@anvers.com</schema:email>
     </ns1:ParkingManager>
   </ns1:parkingManager>

   <schema:contactPoint>
     <rdf:Description>
       <schema:contactType>Emergency Contact Point</schema:contactType>
       <schema:availableLanguage rdf:resource="http://id.loc.gov/
           vocabulary/iso639-1/fr"/>
       <schema:availableLanguage rdf:resource="http://id.loc.gov/
           vocabulary/iso639-1/en"/>
       <schema:telephone>0754234448</schema:telephone>
     </rdf:Description>
   </schema:contactPoint>

   <schema:address>
     <schema:PostalAddress>
```

```
35       <schema:streetAddress>41 Boulevard de Rochechouart</schema:
             streetAddress>
36       <schema:postalCode>75009</schema:postalCode>
37       <schema:addressLocality>Paris, France</schema:addressLocality>
38      </schema:PostalAddress>
39    </schema:address>
40
41  </rdf:Description>
42
43 </rdf:RDF>
```

Listing A.1: Parking Dataset using DCAT Vocabulary

A.2 Parking Example JSON Listing

```
1 [{"@id":"_:b0","http://schema.org/contactType":[{"@value":"Emergency
      Contact Point"}],"http://schema.org/availableLanguage":[{"@id":"http
      ://id.loc.gov/vocabulary/iso639-1/fr"},{"@id":"http://id.loc.gov/
      vocabulary/iso639-1/en"}],"http://schema.org/telephone":[{"@value
      ":"0754234448"}]},{"@id":"_:b1","@type":["http://schema.org/
      PostalAddress"],"http://schema.org/streetAddress":[{"@value":"41
      Boulevard de Rochechouart"}],"http://schema.org/postalCode":[{"@value
      ":"75009"}],"http://schema.org/addressLocality":[{"@value":"Paris,
      France"}]},{"@id":"http://example.org/data/Anvers","@type":["http://
      example.org/ontology/ParkingManager"],"http://purl.org/goodrelations/
      v1#legalName":[{"@value":"Anvers Parking"}],"http://schema.org/
      telephone":[{"@value":"0754234448"}],"http://schema.org/email":[{"
      @value":"parking@anvers.com"}]},{"@id":"http://example.org/data/
      Anvers4","@type":["http://example.org/ontology/ParkingFacility"],"
      http://purl.org/dc/terms/title":[{"@value":"Anvers"}],"http://www.
      opengis.net/ont/geosparql#lat":[{"@value":"48.83523833495664","@type
      ":"http://www.w3.org/2001/XMLSchema#decimal"}],"http://www.opengis.
      net/ont/geosparql#long":[{"@value":"2.37322158810141","@type":"http://
      www.w3.org/2001/XMLSchema#decimal"}],"http://example.org/ontology/
      carParkingSpaces":[{"@value":100}],"http://example.org/ontology/
      motocycleParkingSpaces":[{"@value":50}],"http://example.org/ontology/
      parkingManager":[{"@id":"http://example.org/data/Anvers"}],"http://
      schema.org/contactPoint":[{"@id":"_:b0"}],"http://schema.org/address
      ":[{"@id":"_:b1"}]},{"@id":"http://example.org/ontology/
      ParkingFacility"},{"@id":"http://example.org/ontology/ParkingManager
      "},{"@id":"http://id.loc.gov/vocabulary/iso639-1/en"},{"@id":"http://
      id.loc.gov/vocabulary/iso639-1/fr"},{"@id":"http://schema.org/
      PostalAddress"}]
```

Listing A.2: Parking Dataset using DCAT Vocabulary

Appendix B

LDP-DL Concrete Syntax

B.1 LDP-DL Vocabulary

In this section, we describe the LDP-DL vocabulary and its different components that are shown in Figure B.1. The namespace for this vocabulary is `https://w3id.org/ldpdl/ns#` and the prefix used for it is "`ldl`".

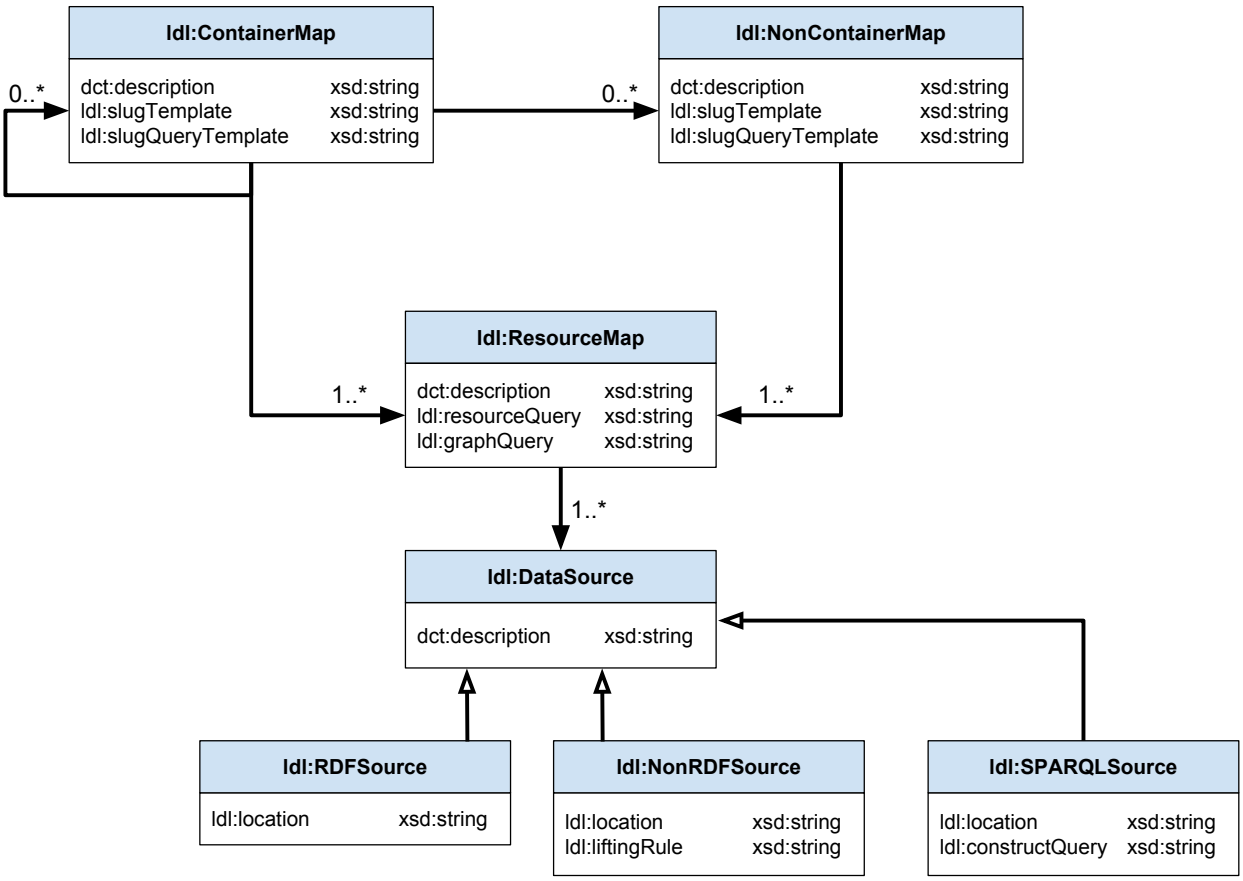

Figure B.1: UML Class Diagram of LDP-DL's Concrete Model

We describe classes and properties from th above model in Appendix B.1.1 and Appendix B.1.2 below.

B.1.1 Classes

ldl:ContainerMap describes basic containers, their RDF graph and members. Constraints on instances of `ldl:ContainerMap`:

- MAY have exactly one value specified by the property `ldl:slugTemplate` or

ldl:NonContainerMap describes non-containers, LDP RDF sources that are not containers, and their RDF graph. Constraints on instances of `ldl:ContainerMap`:

ldl:ResourceMap specifies a rule for generating a set of related resources and their RDF graphs. Constraints on instances of ldl:ResourceMap:

ldl:DataSource the data source from where RDF resources and their description can be obtained. ldl:DataSource is an abstract class. It has three subclasses:

- **`ldl:RDFSource`**: describes an RDF data source
- **`ldl:NonRDFSource`**: describes a Non-RDF data source
- **`ldl:SPARQLSource`**: describes an RDF Graph with respect to a SPARQL endpoint

B.1.2 Properties

ldl:slugTemplate provides a string that MAY be used when generating the IRI of a LDP-RS

- domain: ldl:ContainerMap or ldl:NonContainerMap
- range: `xsd:string`

ldl:slugQueryTemplate specifies a query pattern to generate a string which acts a slug.

- domain: ldl:ContainerMap or ldl:NonContainerMap
- range: `xsd:string` having the following constraints:
  - It MUST encode a SPARQL query pattern as defined in SPARQL 1.1 Query Language
  - The related resource variable is already binded with the IRI of a related resource
  - Some parent variables are with already binded with the parents of a related resource.
  - The results of evaluating the SPARQL query pattern on an RDF graph MUST be projected to the slug variable

ldl:location specifies the location of an ldl:DataSource

- domain: ldl:RDFSource or ldl:NonRDFSource or ldl:SPARQLSource
  - If the domain is ldl:RDFSource, then the value specified by ldl:location is the URL of the RDFSource
  - if its domain is ldl:SPARQLSource, then the value specified by ldl:location is the URL of the SPARQL endpoint
- range: `xsd:string`

ldl:liftingRule specifies the URL of a lifting rule

- domain: ldl:RDFSource
- range: rdfs:Literal where the literal value is a URL in conformance with RFC 1738

ldl:resourceQuery specifies a query pattern for selecting a set of related resources from a data source

- domain: ldl:ResourceMap
- range: rdfs:Literal having the following constraints:
  - It MUST encode a SPARQL query pattern as defined in SPARQL 1.1 Query Language
  - Some parent variables are with already binded with the parents of a related resource.
  - The results of evaluating the SPARQL query pattern on an RDF graph MUST be projected to the related resource variable
  - The evaluation of the SPARQL query pattern on an RDF graph is based on set semantics

ldl:graphQuery

- domain: ldl:ResourceMap
- range: rdfs:Literal having the following constraints:
  - It MUST encode a (SPARQL CONSTRUCT query)
  - Some variables used in the SPARQL CONSTRUCT query have the following constraints:
    - The related resource variable is already binded with the IRI of a related resource
    - Some parent variables are with already binded with the parents of a related resource.

ldl:dataSource specifies the ldl:DataSource of a ldl:ResourceMap

- domain: ldl:ResourceMap
- range: ldl:DataSource

ldl:resourceMap specifies the ldl:ResourceMap of either a ldl:ContainerMap or ldl:NonContainerMap

- domain: ldl:ContainerMap or ldl:NonContainerMap
- range: ldl:ResourceMap

ldl:containerMap specifies a child ldl:ContainerMap of a ldl:ContainerMap

- domain: ldl:ContainerMap
- range: ldl:ContainerMap

ldl:nonContainerMap

- domain: ldl:ContainerMap
- range: ldl:NonContainerMap

B.1.3 Reserved Variable

Some LDP-DL components require using some properties to specificy SPARQL CONSTRUCT queries or query patterns that contain reserved variables. Like all variables in SPARQL 1.1, they have a name and are prefixed with ? or $ to mark them as variables. LDP-DL reserved variables have a name but to distinguish them from other normal variables, they MUST be marked using ?{} where the variable name appears in between the opening and closing braces like ?{res}, ?{slug} or ?{parent}. These variables are further described in the sections below.

Related Resource Variable The related resource variable represent a related resource. Its name is res and MUST always be used as ?{res}

Ancestor Variables The parent variables set is an infinite set of variables having names defined by the regular expression ?parent(.parent)* (e.g. ?{parent} or ?{parent.parent}). The bindings of parent variables MUST be relative to the parents of a particular related resource. The parents of a related resource is the list of all related resources of ancestor LDPRs starting from and including the direct parent LDPR of a related resource's LDPR. There is a partial one-to-one mapping between parent variables and parents of a particular related resource. Given a particular list of parents for a related resource, the binding of a parent variable is the ith element in the list where i is an integer obtained by counting the number of times the string parent appears in the variable. So ?parent refers to the first element and ?parent.parent refers to the second element and so forth.

Slug Variable The slug variable has name slug and MUST appear only in the value specified by the property ldl:slugQueryTemplate as ?{slug}.

B.2 Mapping from Abstract to Concrete Syntax

We use the the definitions of design documents, `ContainerMaps`, `NonContainerMaps`, `ResourceMaps` and `DataSources` as given in Chapter 4 (Section 4.2.2).

Let:

- DD be a design document
- CM be the set of `ContainerMaps`
- NM be the set of `NonContainerMaps`
- RM be the set of `ResourceMaps`
- DS be the set of `DataSources`
- G be the set of RDF graphs

We use $f_{lk} : \mathbf{IRI} \times \mathbf{IRI} \times \mathbf{IRI} \rightarrow \mathbf{IRI} \times \mathbf{IRI} \times \mathbf{IRI}$ to link different LDP-DL constructs such as `DataSources` to `ResourceMaps`, `NonContainerMaps` to `ContainerMaps`, etc.

$$f_{lk}(i_1, i_2, i_3) = \begin{cases} (i_1, i_2, i_3), & \text{if } i_1 \neq \varnothing \\ \varnothing, & \text{if } i_1 = \varnothing \end{cases}$$

Mapping of a Design Document δ to RDF Graphs $f_\delta : DD \rightarrow G$ defines the mapping between design documents and RDF graph describing them. Given a design document $\delta = (CM, NM)$, the RDF graph describing it is given by:

$$f_\delta(\delta) = \bigcup_{cm \in CM} f_{cm}(\varnothing, cm) \cup \bigcup_{nm \in NM} f_{nm}(\varnothing, nm)$$

Mapping of NonContainerMaps to RDF Graphs $f_{nm} : \mathbf{IRI} \times NM \rightarrow G$ defines the mapping between `NonContainerMaps` and RDF graph describing them. The RDF graph describing a `NonContainerMap` $nm = \langle u_{\mathsf{nm}}, RM \rangle$ is given by:

$$f_{nm}(u_{\mathsf{cm}}, nm) = \{(u_{\mathsf{nm}}, \texttt{rdf:type}, \texttt{ldl:NonContainerMap})\} \cup \bigcup_{rm \in RM} f_{rm}(u_{\mathsf{nm}}, rm)$$
$$\cup f_{lk}(u_{\mathsf{cm}}, \texttt{ldl:nonContainerMap}, u_{\mathsf{nm}})$$

Mapping of ContainerMaps to RDF Graphs $f_{cm} : \mathbf{IRI} \times CM \rightarrow G$ defines the mapping between `ContainerMaps` and RDF graph describing them. The RDF

graph describing a ContainerMap $cm = \langle u_{\mathsf{cm}}, RM, CM, NM \rangle$ is given by:

$$f_{cm}(u_{cm'}, cm) = \{(u_{\mathsf{cm}}, \texttt{rdf:type}, \texttt{ldl:ContainerMap})\} \cup \bigcup_{rm \in RM} f_{rm}(u_{\mathsf{cm}}, rm) \cup \bigcup_{nm \in NM} f_{nm}(u_{\mathsf{cm}}, nm) \cup \bigcup_{cm' \in CM} f_{cm}(u_{\mathsf{cm}}, cm') \cup f_{lk}(u_{cm'}, \texttt{ldl:containerMap}, u_{\mathsf{cm}})$$

Mapping of ResourceMaps to RDF Graphs $f_{rm} : \mathbf{IRI} \times RM \rightarrow G$ defines the mapping between ResourceMaps and RDF graph describing them. The RDF graph describing a ResourceMap $rm = \langle u_{\mathsf{rm}}, \mathsf{qp}, \mathsf{cq}, DS \rangle$ is given by:

$$f_{rm}(i_{ncm}, rm) = \{(u_{\mathsf{rm}}, \texttt{rdf:type}, \texttt{ldl:ResourceMap}), (u_{\mathsf{rm}}, \texttt{ldl:resourceQuery}, \mathsf{qp}), (u_{\mathsf{rm}}, \texttt{ldl:graphQuery}, \mathsf{cq}), (i_{ncm}, \texttt{ldl:resourceMap}, u_{\mathsf{rm}})\} \cup \bigcup_{ds \in DS} f_{ds}(u_{\mathsf{rm}}, ds)$$

Mapping of DataSources to RDF Graphs $f_{ds} : \mathbf{IRI} \times DS \rightarrow G$ defines the mapping between DataSources and RDF graphs describing them. As mentioned in Chapter 4 (Section 4.2.2), we describe only two forms of a DataSource namely $\langle u_{\mathsf{ds}}, u_{\mathsf{loc}} \rangle$ and $\langle u_{\mathsf{ds}}, u_{\mathsf{loc}}, u_{\mathsf{lr}} \rangle$ having the RDF graphs $f_{ds}(\langle u_{\mathsf{ds}}, u_{\mathsf{loc}} \rangle)$ and $f_{ds}(\langle u_{\mathsf{ds}}, u_{\mathsf{loc}}, u_{\mathsf{lr}} \rangle)$ that we further define below.

$$f_{ds}(u_{\mathsf{rm}}, \langle u_{\mathsf{ds}}, u_{\mathsf{loc}} \rangle) = \{(u_{\mathsf{ds}}, \texttt{rdf:type}, \texttt{ldl:RDFSource}), (u_{\mathsf{ds}}, \texttt{ldl:location}, u_{\mathsf{loc}}), (u_{\mathsf{rm}}, \texttt{ldl:dataSource}, u_{\mathsf{ds}})\}$$

$$f_{ds}(u_{\mathsf{rm}}, \langle u_{\mathsf{ds}}, u_{\mathsf{loc}}, u_{\mathsf{lr}} \rangle) = \{(u_{\mathsf{ds}}, \texttt{rdf:type}, \texttt{ldl:NonRDFSource}), (u_{\mathsf{ds}}, \texttt{ldl:location}, u_{\mathsf{loc}}), (u_{\mathsf{ds}}, \texttt{ldl:liftingRule}, u_{\mathsf{lr}}), (u_{\mathsf{rm}}, \texttt{ldl:dataSource}, u_{\mathsf{ds}})\}$$

B.3 Mapping from Concrete to Abstract Syntax

We use the the definitions of design documents, ContainerMaps, NonContainerMaps, ResourceMaps and DataSources as given in Chapter 4 (Section 4.2.2).

Let:

- DD be a design document
- CM be the set of ContainerMaps
- NM be the set of NonContainerMaps
- RM be the set of ResourceMaps
- DS be the set of DataSources
- G be the set of RDF graphs

Mapping of RDF Graphs to Design Documents $f_{g\delta} : \mathbf{IRI} \times G \to DD$ defines the mapping between RDF graphs and design documents. Given an RDF Graph g_δ its mapping to a design document (CM, NM) is given by:

$$f_{g\delta}(g_\delta) = (CM, NM)\ s.t.\ CM = \{f_{gcm}(u_{\mathsf{cm}}, g_\delta) \mid (u_{\mathsf{cm}}, \texttt{rdf:type}, \texttt{ldl:ContainerMap}) \in g_\delta \wedge \nexists u_{cm'}, (u_{cm'}, \texttt{ldl:containerMap}, u_{\mathsf{cm}}) \in g_\delta\},$$
$$NM = \{f_{gnm}(u_{\mathsf{nm}}, g_\delta) \mid (u_{\mathsf{nm}}, \texttt{rdf:type}, \texttt{ldl:NonContainerMap}) \in g_\delta \wedge \nexists u_{cm'}, (u_{cm'}, \texttt{ldl:nonContainerMap}, u_{\mathsf{nm}}) \in g_\delta\}$$

Mapping of RDF Graphs to NonContainerMaps $f_{gnm} : \mathbf{IRI} \times G \to NM$ defines the mapping between RDF graphs and `NonContainerMaps`. Given an RDF Graph g_δ its mapping to a `NonContainerMap` $nm = (u_{\mathsf{cm}}, RM)$ is given by:

$$f_{gnm}(u_{\mathsf{nm}}, g_\delta) = (u_{\mathsf{nm}}, RM)\ s.t.\ RM = \{f_{grm}(u_{\mathsf{rm}}, g_\delta) \mid (u_{\mathsf{nm}}, \texttt{ldl:resourceMap}, u_{\mathsf{rm}}) \in g_\delta\}$$

Mapping of RDF Graphs to ContainerMaps $f_{gcm} : \mathbf{IRI} \times G \to CM$ defines the mapping between RDF graphs and `ContainerMaps`. Given an RDF Graph g_δ its mapping to a `NonContainerMap` $nm = (u_{\mathsf{cm}}, RM, CM, NM)$ is given by:

$$f_{gcm}(u_{\mathsf{cm}}, g_\delta) = (u_{\mathsf{cm}}, RM, CM, NM)\ s.t$$
$$RM = \{f_{grm}(u_{\mathsf{rm}}, g_\delta) \mid (u_{\mathsf{cm}}, \texttt{ldl:resourceMap}, u_{\mathsf{rm}}) \in g_\delta\},$$
$$CM = \{f_{gcm}(u_{cm'}, g_\delta) \mid (u_{\mathsf{cm}}, \texttt{ldl:containerMap}, u_{cm'}) \in g_\delta\},$$
$$NM = \{f_{gnm}(u_{nm}, g_\delta) \mid (u_{\mathsf{cm}}, \texttt{ldl:nonContainerMap}, u_{\mathsf{nm}}) \in g_\delta\}$$

Mapping of RDF Graphs to ResourceMap $f_{grm} : \mathbf{IRI} \times G \to RM$ defines the mapping between RDF graphs and `ResourceMaps`. Given an RDF Graph g_δ its mapping to a `ResourceMap` $rm = (u_{\mathsf{rm}}, DS)$ is given by:

$$f_{grm}(u_{\mathsf{rm}}, g_\delta) = (u_{\mathsf{rm}}, DS)\ s.t.\ DS = \{f_{gds}(u_{\mathsf{ds}}, g_\delta) \mid (u_{\mathsf{rm}}, \texttt{ldl:dataSource}, u_{\mathsf{ds}}) \in g_\delta\}$$

Mapping of RDF Graphs to DataSources $f_{gds} : \mathbf{IRI} \times G \to DS$ defines the mapping between RDF graphs and `DataSources`. Given an RDF Graph g_δ its mapping to a `DataSource` is given by:

$$f_{gds}(u_{\mathsf{ds}}, g_\delta) = \begin{cases} (u_{\mathsf{ds}}, u_{\mathsf{loc}}), & \text{if } (u_{\mathsf{ds}}, \texttt{ldl:location}, u_{\mathsf{loc}}) \in g_\delta \wedge (u_{\mathsf{ds}}, \texttt{ldl:liftingRule}, u_{\mathsf{lr}}) \notin g_\delta \\ (u_{\mathsf{ds}}, u_{\mathsf{loc}}, u_{\mathsf{lr}}) & \text{if } \{(u_{\mathsf{ds}}, \texttt{ldl:location}, u_{\mathsf{loc}}), (u_{\mathsf{ds}}, \texttt{ldl:location}, u_{\mathsf{loc}})\} \subseteq g_\delta \end{cases}$$

Appendix C

Materials for LDP-DL

In this appendix, we provide supplementary materials for the description of LDP-DL given in Chapter 4

C.1 Example LDP-DL design

In Chapter 4 (Figure 4.2), we provided an LDP-DL design without the query patterns and CONSTRUCT queries. The LDP-DL design with these details are shown in Figure C.1.

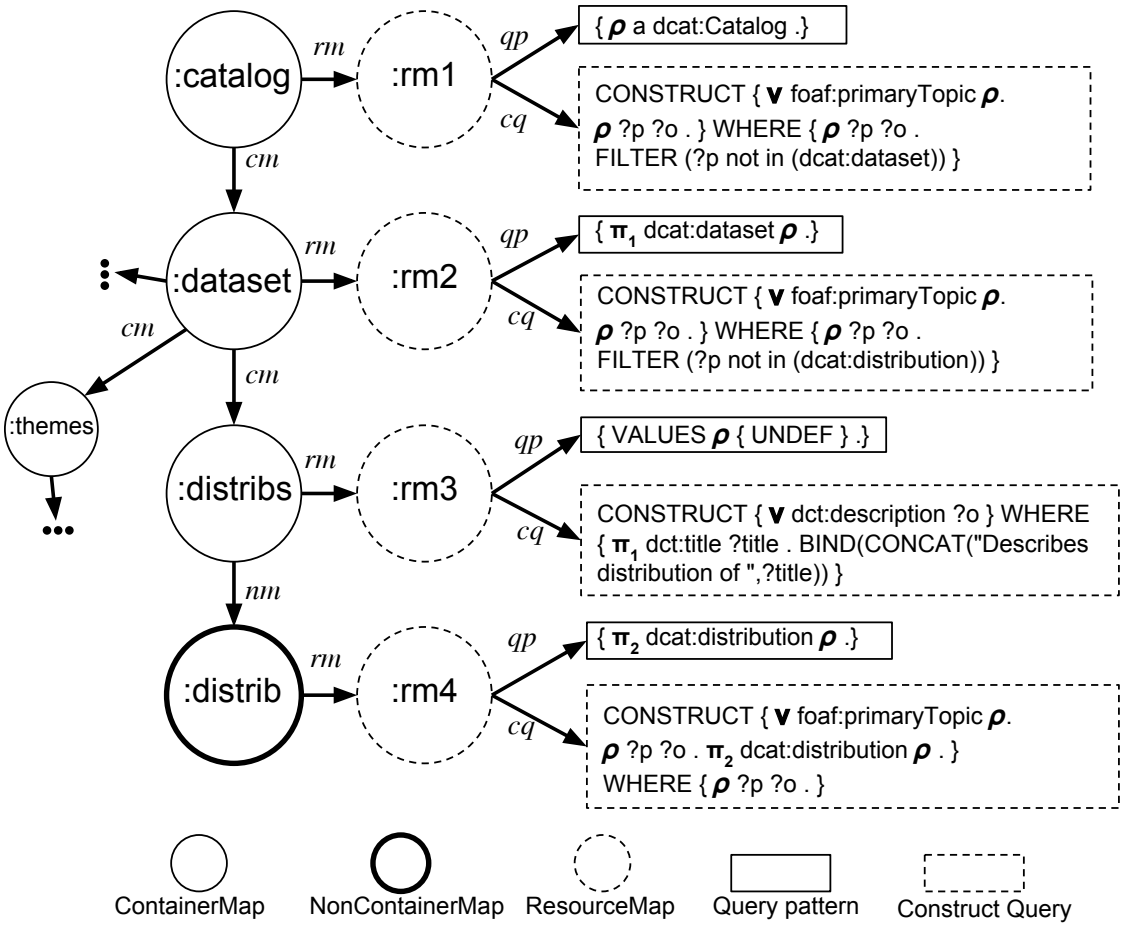

FIGURE C.1: Example of an LDP-DL design in the abstract syntax.

C.1.1 Illustrating Example Using Abstract Syntax Concepts

In this section, we describe the evaluation of the LDP-DL design in Figure C.1 using concepts from the abstract syntax given in Chapter 4 (Section 4.2.2). We informally explains the semantics of LDP-DL by describing the satisfaction of its constructs using a concrete example of the design document in Figure C.1. The latter design document is used for building the LDP[1] having the structure shown in Figure 3.8 (b) using the data source in Figure 3.8 (a). Also, whenever we refer to a resource having the prefix ex, dex or :, then it should be assumed that reference is being made to a resource either from Figure 3.8 (a), Figure 3.8 (b) or Figure C.1 respectively.

As mentioned before, all the `ResourceMaps` in the design document have a `DataSource` `:dataSource`. Therefore, we start the informal description by describing the satisfaction of `:dataSource`.

:dataSource

As mentioned before, we consider only two forms of a `DataSource`. Therefore, abstractly, `:dataSource` can be seen as being either $\langle u_{ds_1}, u_{loc_1} \rangle$ or $\langle u_{ds_1}, u_{loc_1}, u_{lr_1} \rangle$. In the former case, `:dataSource` is satisfied if its interpretation, that is $\mathcal{I}_{\mathcal{S}}(u^{\mathcal{I}}_{ds1})$, is the RDF graph located at u_{loc_1}. In the latter case, `:dataSource` is satisfied if $\mathcal{I}_{\mathcal{S}}(u^{\mathcal{I}}_{ds1})$ is the RDF graph obtained by executing the lifting rule found at u_{lr_1} on the document found at u_{loc_1}.

In either case, in the rest of the informal description, we consider the RDF graph obtained to be the one in Figure 3.8 (a). Now that we have described the satisfaction of `:dataSource`, we proceed with that of `:catalog`.

:catalog

At the top level of the design document, the `ContainerMap` `:catalog` uses the `ResourceMap` `:rm1` to generate the top level containers. At this level, `:rm1` is evaluated with an empty ancestor list. Using its query pattern, related resources extracted from the source are DCAT catalogs `ex:paris-catalog` and `ex:toulouse-catalog`. For each of them, an IRI is generated, namely `dex:paris-catalog` and `dex:toulouse-catalog`. Also, to satisfy the map `:rm1`, the RDF graph associated with the container IRI is obtained using its `CONSTRUCT` query, where the variable ρ is bound to the related resource IRI, and ν to the IRI of the new LDP resource. For example, when doing so for `dex:paris-catalog`, ρ is bound to `ex:paris-catalog`, and ν to `dex:paris-catalog`. Finally, new containers generated from `:catalog` must define their members as well, and is thus satisfied only if their members correspond to the resources generated by their underlying `ContainerMaps` and `NonContainerMaps` which in this case, is `:dataset` only.

:dataset

The `ContainerMap` `:dataset` is used to generate members for containers generated from `:catalog`. Let us consider the case for `dex:paris-catalog`. Its related resource is `ex:paris-catalog` and its members must only have related resources that are

[1] http://opensensingcity.emse.fr/ldpdfend/catalogs/ldp

DCAT datasets from this catalog. This is why the extraction of these resources is parameterized by the parent variable π_1 in the query pattern :rm2. π_1 is binded to the first element of the ancestor list which at this stage is ex:paris-catalog. The evaluation of :dataset generates two containers, dex:parking and dex:busStation, that are added to the members of dex:paris-catalog. The satisfaction of :dataset also depends on its underlying ContainerMaps and NonContainerMaps. In this case, we consider only :distribs and continue our explanation using its evaluation.

:distribs

The map :distribs is used to generate members for containers generated by :dataset. Consider the case of doing so for dex:parking whose related resource is ex:parking. In this context, the aim of using :distribs is to generate a container to describe the set of distributions of ex:parking. Note that in the data source, there is no explicit resource to describe this set. This is why, in the ResourceMap :rm3, the query pattern returns a single result where ρ is unbound. Although the query pattern does not use any ancestor variable, it is evaluated using the ancestor list (ex:paris-catalog,ex:parking) and thus ancestor variables π_1 and π_2 are bound. The evaluation of :distribs in the context of dex:parking generates a single container dex:pDistributions. :distribs is satisfied when a single container is generated without a related resource and when its underlying ContainerMaps and NonContainerMaps, which in this case is :distrib, is satisfied.

:distrib

Finally, the NonContainerMap :distrib is used to generate non-containers for each distribution of a DCAT dataset. Consider the case of doing so for dex:pDistributions with ancestor list (ex:paris-catalog,ex:parking,$\emptyset$). In this context, the proper related resource that must be used to extract the relevant distributions is associated with the grand parent container. This is why the query pattern of :rm4 uses π_2, bound to ex:parking, rather than π_1. Using the result (ex:pJSON and ex:pCSV) of this query pattern, two non-containers dex:pJSON and dex:pCSV in dex:pDistributions are generated using :distrib. In general, any ancestor's related resources can be referenced through the ancestor variables π_i simultaneously, even when they are unbound.

C.2 Static/Dynamic LDP Dataset

In this section, we describe the generation of static/dynamic LDP datasets from a design document and the generation of LDP dataset from static/dynamic LDP datasets. First, we do so for static LDP dataset, then for dynamic LDP dataset.

C.2.1 Static LDP Dataset

In Section 4.4.1, we provided a family of algorithms for generating an LDP dataset from a design document. The generation of static/dynamic LDP dataset is very similar to the generation of LDP datasets. To generate static LDP datasets, the same

algorithms can be used with the exception that the function $geniri$ in Algorithm 1 must generate temporary identifiers for LDP resources.

Then, to generate an LDP dataset from a static LDP dataset, all temporary identifiers in the static LDP dataset should be replaced by the final IRIs of LDP resources. Algorithm 9 shows an example of an algorithm that generates an LDP dataset (Σ) from a static LDP dataset (ϕ). IdToIRI is mappings of temporary identifiers of LDP resources, occurring in ϕ, to their final IRI. Assume that IdToIRI(id) is the IRI for a particular temporary identifier id.

Algorithm 9 Generation of LDP dataset from Static LDP dataset

```
1:  procedure GenLDPDatasetS(φ,IdToIRI)
2:      Σ ← ∅ //Initialize the LDP dataset
3:      for all (id, g) ∈ φ) do
4:          Σ ← Σ∪(IdToIRI(id),g) //Replace temporary identifiers for non-containers
5:      end for
6:      for all (id, g, M) ∈ φ) do
7:          Σ ← Σ∪ (IdToIRI(id),g,M)//Replace temporary identifiers for containers
8:      end for
9:      return Σ
10: end procedure
```

C.2.2 Dynamic LDP dataset

Overall, when generating a dynamic LDP dataset, instead of evaluating the CONSTRUCT query of a `ResourceMap` to obtain the graph of LDP resources, the CONSTRUCT with its variables replaces by the proper mappings is saved and later use to generate the graph of the LDP resource. Therefore, all the algorithms for generating LDP datasets can be used with the exception of that for processing `ResourceMaps` (Algorithm 1). We provide the algorithm for processing `ResourceMaps` when generating dynamic LDP dataset in Algorithm 10.

In Algorithm 10, we assume the existence of a function $replace(cq, mappings)$ that takes a CONSTRUCT query cq and a set of variable mappings $mappings$ and replaces a variable in cq by its mapping if there is a mapping for that variable in $mappings$.

When generating a dynamic LDP dataset, the function $replace$ (Algorithm 10, Line 14) is used to replace the reserved variables in the CONSTRUCT query of a `ResourceMap` by their mappings. Then, on Line 15, the CONSTRUCT query is then used to create a `GraphDescription` (Definition 18).

An LDP dataset is generated from a dynamic LDP dataset at a particular time instant by evaluating its dynamic container and non-container structures. The algorithm for doing so in provided in Algorithm 11 and we assume the existence of the following functions in it:

- $members(x)$: returns the set of members from the dynamic container x;
- $gDes(x)$: return a `GraphDescription` from the structure x.

Algorithm 11 involve the evaluation of dynamic container structures (Algorithm 11, Line 3 - 7) and dynamic non-container structures that involve the evaluation of `GraphDescriptions`. As mentioned before, a `GraphDescription` (cq, DS)

Algorithm 10 Evaluation of a ResourceMap rm

1: **procedure** $Eval_{rm}(rm = (u_{\mathsf{rm}}, \mathsf{qp}, \mathsf{cq}, \mathbf{DS}), \vec{p})$
2: $result \leftarrow \varnothing$
3: $g_s \leftarrow [\![\mathbf{DS}]\!]_{source}$//perform merge of all RDF graph obtain from every data source
4: **for all** $i \in (1..len(\vec{p}))$ **do**//bind ancestor variables
5: $\mu_{\vec{p}}(\pi_i) = \vec{p}[i]$
6: **end for**
7: $\Omega \leftarrow \Pi_\rho([\![\mathsf{qp}]\!]_{g_s} \bowtie \{\mu_{\vec{p}}\})$ //get related resources
8: $result \leftarrow \varnothing$ //structure to hold new resources generated from rm
9: **for all** $\mu \in \Omega$ **do**//Iterate over all mappings
10: **if** $\mu(\rho) \neq \epsilon$ **then** //when there exist a related resource
11: $\mu_\rho(\rho) = r$
12: **end if**
13: $n \leftarrow geniri(\mu(\rho), \dots)$ //generate IRI of new resource
14: $graphQuery \leftarrow replace(\mathsf{cq}, \{\{\nu \leftarrow n\}, \{\mu_\rho\}, \{\mu_{\vec{p}}\}\})$/*replace reserved variables in CONSTRUCT query*/
15: $graphDescription \leftarrow (graphQuery, \mathbf{DS})$//create graph description
16: $result \leftarrow result \cup \{(n, \mu(\rho), graphDescription)\}$
17: **end for**
18: return $result$
19: **end procedure**

consist of a CONSTRUCT query cq and a set of `DataSources` DS. To evaluate the latter structure, the RDF merge of all the individual data sources in DS is performed to generate an RDF graph on which cq is evaluated to generate either the graph of the container or non-container. Finally, all containers (Algorithm 11, Line 11) and non-containers (Algorithm 11, Line 6) are added to the LDP dataset Σ.

Algorithm 11 Generation of an LDP dataset from a dynamic LDP dataset

1: **procedure** $Eval_\phi(\phi)$
2: $\Sigma \leftarrow \varnothing$ Initialize LDP Dataset
3: **for all** $dnc \in dNC$ **do**Evaluate dynamic non-containers
4: $crm \leftarrow gDes(dnc)$ dnc has the form (n, crm)
5: $g \leftarrow Eval_{gdes}(gdes)$
6: $AddNonContainer(\Sigma, (iri(dnc), g))$
7: **end for**
8: **for all** $dc \in dC$ **do**Evaluate dynamic containers
9: $gdes \leftarrow gDes(dc)$ dc has the form (n, crm, M)
10: $g \leftarrow Eval_{gdes}(gdes)$
11: $AddContainer(\Sigma, (iri(dnc), g, members(dc)))$
12: **end for**
13: return Σ return the LDP dataset
14: **end procedure**

Appendix D

Materials for Evaluation

Below, we provide the files that we used in our evaluation and provide links to the generated LDPs in it.

D.1 Random DCAT Dataset Generation

In Chapter 6 (Section 6.1.1), we briefly describe how we generated the RDF graphs for the performance test. Here, we go more in details about the Python program, shown in Listing D.1, used generating these RDF graphs. In it, there two variables `top` (Listing D.1, Line 5) and `placeHolder` (Listing D.1, Line 21). The `top` variable contains the definition of a DCAT catalog that is made of 6 triples. The `placeHolder` definition of a DCAT dataset together with its distributions that is made up of 37 triples.

The DCAT dataset and its distributions from the placeholder have a string `<theRandomCode>` in their IRIs. To generate a random DCAT dataset and its distributions, the string `<theRandomCode>` is replaced by a 8-bit word generated using a random number generator. Then, an RDF graph is obtained by concatenating a DCAT catalog with several DCAT dataset.

To generate the RDF graph of one million triple, at least 27027 DCAT datasets and distributions need to be added. Therefore, we make the program iterates for 27050 times and on every 50 iterations, an RDF graph is serialized containing the original DCAT catalog and incrementally all the DCAT datasets generated so far. For example, on the 50^{th} run, the RDF graph is serialized containing the DCAT catalog and 50 DCAT datasets and their distributions. Then, on the 100^{th} run, another RDF graph is serialized with the DCAT catalog and 100 DCAT datasets and their distributions, 50 of which comes from the first 50 iterations.

In total, we generate 541 RDF graphs with the smallest containing 1856 triples that is made up of one DCAT catalog and 50 DCAT dataset and the biggest one contains 1000856 triples and is made up of one DCAT catalog and 27050 DCAT datasets.

```
1 # coding=utf-8
2 from rdflib import Graph
3 import random
4
5 top = """@prefix dcat: <http://www.w3.org/ns/dcat#> .
6 @prefix dct: <http://purl.org/dc/terms/> .
7 @prefix foaf: <http://xmlns.com/foaf/0.1/> .
8 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
9 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

```
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<https://example.com/api/v2/catalog/exports/ttl> a dcat:Catalog ;
dct:Language <http://id.loc.gov/vocabulary/iso639-1/en>,
<http://id.loc.gov/vocabulary/iso639-1/fr> ;
dct:description "Bistrotdepays Catalog" ;
dct:publisher <http://www.opendatasoft.com> ;
dct:title "Bistrotdepays's catalog" . """

placeHolder = """<https://example.com/api/v2/catalog/exports/ttl> dcat:
    dataset  <https://example.com/api/v2/catalog/datasets/<theRandomCode>/
    animations_bistrots_de_pays> .

<https://example.com/api/v2/catalog/datasets/<theRandomCode>/
    animations_bistrots_de_pays> a dcat:Dataset ;
dct:description "Liste des animations dans les Ãl'tablissements du label
    Bistrot de Pays." ;
dct:identifier "animations_bistrots_de_pays" ;
dct:language <http://id.loc.gov/vocabulary/iso639-1/fr> ;
dct:publisher [ a foaf:Agent ;
rdfs:label "Example datasets" ] ;
dct:title "example datasets" ;
dcat:distribution <https://example.com/api/v2/catalog/datasets/<
    theRandomCode>/animations_bistrots_de_pays-csv>,
<https://example.com/api/v2/catalog/datasets/<theRandomCode>/
    animations_bistrots_de_pays-geojson>,
<https://example.com/api/v2/catalog/datasets/<theRandomCode>/
    animations_bistrots_de_pays-json>,
<https://example.com/api/v2/catalog/datasets/<theRandomCode>/
    animations_bistrots_de_pays-shp> .

<https://example.com/api/v2/catalog/datasets/<theRandomCode>/
    animations_bistrots_de_pays-csv> a dcat:Distribution ;
dct:description "csv export of https://example.com/api/v2/catalog/
    datasets/animations_bistrots_de_pays" ;
dct:format "csv" ;
dct:license "http://opendatacommons.org/licenses/odbl/1.0/" ;
dcat:accessURL <https://example.com/api/v2/catalog/datasets/
    animations_bistrots_de_pays/exports/csv> ;
dcat:mediaType "text/csv" .

<https://example.com/api/v2/catalog/datasets/<theRandomCode>/
    animations_bistrots_de_pays-geojson> a dcat:Distribution ;
dct:description "geojson export of https://example.com/api/v2/catalog/
    datasets/animations_bistrots_de_pays" ;
dct:format "geojson" ;
dct:license "http://opendatacommons.org/licenses/odbl/1.0/" ;
dcat:accessURL <https://example.com/api/v2/catalog/datasets/
    animations_bistrots_de_pays/exports/geojson> ;
dcat:mediaType "application/json" .

<https://example.com/api/v2/catalog/datasets/<theRandomCode>/
    animations_bistrots_de_pays-json> a dcat:Distribution ;
dct:description "json export of https://example.com/api/v2/catalog/
    datasets/animations_bistrots_de_pays" ;
dct:format "json" ;
dct:license "http://opendatacommons.org/licenses/odbl/1.0/" ;
dcat:accessURL <https://example.com/api/v2/catalog/datasets/
    animations_bistrots_de_pays/exports/json> ;
dcat:mediaType "application/json" .
```

```
55
56 <https://example.com/api/v2/catalog/datasets/<theRandomCode>/
       animations_bistrots_de_pays-shp> a dcat:Distribution ;
57 dct:description "shp export of https://example.com/api/v2/catalog/
       datasets/animations_bistrots_de_pays" ;
58 dct:format "shp" ;
59 dct:license "http://opendatacommons.org/licenses/odbl/1.0/" ;
60 dcat:accessURL <https://example.com/api/v2/catalog/datasets/
       animations_bistrots_de_pays/exports/shp> ;
61 dcat:mediaType "application/zip" ."""
62
63 ac = ""
64 num = 27050
65 for i in range(1,num+1):
66    print i
67    ran = str(random.getrandbits(30))
68    ac = ac + placeHolder.replace("<theRandomCode>",ran)
69    if (i % 50 == 0):
70       final = top + ac
71 f = open("datasets/"+str(i),"w")
72 f.write(top+ac)
73 f.close()
```

Listing D.1: Python Program for generating Random RDF graphs based on the DCAT vocabulary

D.2 Modular Design Reusability

In *Modular Design Reusability* (Section 6.2.1), we provided a document (Listing 6.3) containing a partial LDP-DL design that refers to instances of LDP-DL constructs defined externally. In this section, we provide these external documents containing the definition of `:cinemasCM` (Listing D.2), `:bibliotheque_mediathequesCM` (Listing D.3), `:chateau_monumentsCM` (Listing D.4) and `:parc_animalier_themesCM` (Listing D.5).

```
1 @prefix : <https://w3id.org/ldpdl/#> .
2 @prefix data: <http://opensensingcity.emse.fr/LDPDesign/data/> .
3 @prefix s:<http://lodpaddle.univ-nantes.fr/> .
4 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
5
6 data:cinemasCM a :NullContainerMap;
7    :slugTemplate "cinemas";
8    :nonContainerMap data:cinemaCM;
9 .
10
11 data:cinemaCM a :NonContainerMap;
12   :slugQueryTemplate "{BIND(str(?{res}) as ?nres) BIND(if(contains(?nres,
         '#'),true,false) as ?hash) BIND(SUBSTR(?nres,STRLEN(?nres),1) as ?
        lstr) BIND(if(?lstr='/',SUBSTR(?nres,0,STRLEN(?nres)),?nres) as ?
        fres) BIND(if(?hash,REPLACE(?fres,'.*#',''),REPLACE(?fres,'.*/','')
        ) as ?template)}";
13   :resourceMap data:cinemaRM;
14 .
15
16 data:cinemaRM a :ResourceMap;
```

```
17  :slugQueryTemplate "{BIND(str(?{res}) as ?nres) BIND(if(contains(?nres,
        '#'),true,false) as ?hash) BIND(SUBSTR(?nres,STRLEN(?nres),1) as ?
        lstr) BIND(if(?lstr='/',SUBSTR(?nres,0,STRLEN(?nres)),?nres) as ?
        fres) BIND(if(?hash,REPLACE(?fres,'.*#',''),REPLACE(?fres,'.*/',''))
         as ?template)}";
18  :resourceQuery "{ ?{res} rdf:type s:cinema .}";
19  :graphQuery "CONSTRUCT { ?{res} ?p ?o . ?s ?p ?{res} . } WHERE { { ?{res
        } ?p ?o . } UNION { ?s ?p ?{res} . } }" .
```

Listing D.2: Document containing partial LDP-DL design describing :cinemaCM

```
1 @prefix : <https://w3id.org/ldpdl/#> .
2 @prefix data: <http://opensensingcity.emse.fr/LDPDesign/data/> .
3 @prefix s:<http://lodpaddle.univ-nantes.fr/> .
4 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
5
6 data:bibliotheque_mediathequesCM a :NullContainerMap;
7    :slugTemplate "bibliotheque_mediatheques";
8    :nonContainerMap data:bibliotheque_mediathequeCM;
9 .
10
11 data:bibliotheque_mediathequeCM a :NonContainerMap;
12   :slugQueryTemplate "{BIND(str(?{res}) as ?nres) BIND(if(contains(?nres,
         '#'),true,false) as ?hash) BIND(SUBSTR(?nres,STRLEN(?nres),1) as ?
        lstr) BIND(if(?lstr='/',SUBSTR(?nres,0,STRLEN(?nres)),?nres) as ?
        fres) BIND(if(?hash,REPLACE(?fres,'.*#',''),REPLACE(?fres,'.*/','')
        ) as ?template)}";
13   :resourceMap data:bibliotheque_mediathequeRM;
14 .
15
16 data:bibliotheque_mediathequeRM a :ResourceMap;
17  :slugQueryTemplate "{BIND(str(?{res}) as ?nres) BIND(if(contains(?nres,
        '#'),true,false) as ?hash) BIND(SUBSTR(?nres,STRLEN(?nres),1) as ?
        lstr) BIND(if(?lstr='/',SUBSTR(?nres,0,STRLEN(?nres)),?nres) as ?
        fres) BIND(if(?hash,REPLACE(?fres,'.*#',''),REPLACE(?fres,'.*/',''))
         as ?template)}";
18  :resourceQuery "{ ?{res} rdf:type s:bibliotheque_mediatheque .}";
19  :graphQuery "CONSTRUCT { ?{res} ?p ?o . ?s ?p ?{res} . } WHERE { { ?{res
        } ?p ?o . } UNION { ?s ?p ?{res} . } }" .
```

Listing D.3: Document containing partial LDP-DL design describing :bibliotheque_mediathequesCM

```
1 @prefix : <https://w3id.org/ldpdl/#> .
2 @prefix data: <http://opensensingcity.emse.fr/LDPDesign/data/> .
3 @prefix s:<http://lodpaddle.univ-nantes.fr/> .
4 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
5
6 data:chateau_monumentsCM a :NullContainerMap;
7    :slugTemplate "Chateaux";
8    :nonContainerMap data:chateau_monumentCM;
9 .
10
11 data:chateau_monumentCM a :NonContainerMap;
12   :slugQueryTemplate "{BIND(str(?{res}) as ?nres) BIND(if(contains(?nres,
         '#'),true,false) as ?hash) BIND(SUBSTR(?nres,STRLEN(?nres),1) as ?
        lstr) BIND(if(?lstr='/',SUBSTR(?nres,0,STRLEN(?nres)),?nres) as ?
        fres) BIND(if(?hash,REPLACE(?fres,'.*#',''),REPLACE(?fres,'.*/','')
        ) as ?template)}";
```

```
13   :resourceMap data:chateau_monumentRM;
14 .
15
16 data:chateau_monumentRM a :ResourceMap;
17   :slugQueryTemplate "{BIND(str(?{res}) as ?nres) BIND(if(contains(?nres,
        '#'),true,false) as ?hash) BIND(SUBSTR(?nres,STRLEN(?nres),1) as ?
        lstr) BIND(if(?lstr='/',SUBSTR(?nres,0,STRLEN(?nres)),?nres) as ?
        fres) BIND(if(?hash,REPLACE(?fres,'.*#',''),REPLACE(?fres,'.*/',''))
         as ?template)}";
18   :resourceQuery "{ ?{res} rdf:type s:chateau_monument .}";
19   :graphQuery "CONSTRUCT { ?{res} ?p ?o . ?s ?p ?{res} . } WHERE { { ?{res
        } ?p ?o . } UNION { ?s ?p ?{res} . } }" .
```

Listing D.4: Document containing partial LDP-DL design describing `:chateau_monumentsCM`

```
1 @prefix : <https://w3id.org/ldpdl/#> .
2 @prefix data: <http://opensensingcity.emse.fr/LDPDesign/data/> .
3 @prefix s:<http://lodpaddle.univ-nantes.fr/> .
4 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
5
6 data:parc_animalier_themesCM a :NullContainerMap;
7    :slugTemplate "parc_animalier_themes";
8    :nonContainerMap data:parc_animalier_themeCM;
9 .
10
11 data:parc_animalier_themeCM a :NonContainerMap;
12   :slugQueryTemplate "{BIND(str(?{res}) as ?nres) BIND(if(contains(?nres,
         '#'),true,false) as ?hash) BIND(SUBSTR(?nres,STRLEN(?nres),1) as ?
        lstr) BIND(if(?lstr='/',SUBSTR(?nres,0,STRLEN(?nres)),?nres) as ?
        fres) BIND(if(?hash,REPLACE(?fres,'.*#',''),REPLACE(?fres,'.*/','')
        ) as ?template)}";
13   :resourceMap data:parc_animalier_themeRM;
14 .
15
16 data:parc_animalier_themeRM a :ResourceMap;
17   :slugQueryTemplate "{BIND(str(?{res}) as ?nres) BIND(if(contains(?nres,
        '#'),true,false) as ?hash) BIND(SUBSTR(?nres,STRLEN(?nres),1) as ?
        lstr) BIND(if(?lstr='/',SUBSTR(?nres,0,STRLEN(?nres)),?nres) as ?
        fres) BIND(if(?hash,REPLACE(?fres,'.*#',''),REPLACE(?fres,'.*/',''))
         as ?template)}";
18   :resourceQuery "{ ?{res} rdf:type s:parc_animalier_theme .}";
19   :graphQuery "CONSTRUCT { ?{res} ?p ?o . ?s ?p ?{res} . } WHERE { { ?{res
        } ?p ?o . } UNION { ?s ?p ?{res} . } }" .
```

Listing D.5: Document containing partial LDP-DL design describing `:parc_animalier_themesCM`

D.3 Dynamic LDP

In *Dynamic LDP* (Section 6.2.2), we used a design document that refers to a static and dynamic non-RDF data source. The design document is shown in Listing D.6. The lifting rule for the static and dynamic data source is shown in Listing D.7 and Listing D.8 respectively.

```
1 @prefix : <http://opensensingcity.emse.fr/LDPDesignVocabulary/> .
```

```
@prefix data: <http://opensensingcity.emse.fr/LDPDesign/data/> .
@prefix dcat: <http://www.w3.org/ns/dcat#>
@prefix pk: <http://opensensingcity.emse.fr/ontologies/parking/> .
@prefix time: <http://www.w3.org/2006/time#> .
@prefix sosa: <http://www.w3.org/ns/sosa/> .

data:parkingsCM a :NullContainerMap;
   :iriTemplate "parkings";
   :nonContainerMap data:parkingCM;
.

data:parkingCM a :NonContainerMap;
   :iriTemplate "$path(res,2)$";
   :resourceMap data:parkingRM;
.

data:parkingRM a :ResourceMap;
   :resourceQuery "{ ?res a pk:ParkingFacility .}";
   :graphQuery "CONSTRUCT { ?res ?p ?o . ?res pk:nbAvailableParkingSpaces
        ?av } WHERE { {SELECT ?p ?o WHERE { ?res ?p ?o . FILTER (?p not
       in (sosa:observation)) } } UNION { SELECT ?av WHERE { ?res sosa:
       observation/pk:nbAvailableParkingSpaces ?av; sosa:observation/time:
       inTimePosition/time:numericPosition ?time } ORDER BY DESC(?time)
       LIMIT 1 } }";
   :dataSource data:DataSource1;
   :dataSource data:DataSource2;
.


data:DataSource1 a :NonRDFSource;
   :location "http://data.metromobilite.fr/api/bbox/json?types=PAR";
   :liftingRule "https://raw.githubusercontent.com/OpenSensingCity/
       DatasetsLiftingRules/master/grenoble/parking/
       grenoble_parking_dynamic.rqg";
.

data:DataSource2 a :NonRDFSource;
   :location "http://data.metromobilite.fr/api/dyn/PAR/json";
   :liftingRule "https://raw.githubusercontent.com/OpenSensingCity/
       DatasetsLiftingRules/master/grenoble/parking/
       grenoble_parking_static.rqg";
.
```

Listing D.6: Design document used in *Dynamic LDP* (Section 6.2.2)

```
BASE <http://example.com/>
PREFIX iter: <http://w3id.org/sparql-generate/iter/>
PREFIX fn: <http://w3id.org/sparql-generate/fn/>
PREFIX pk: <http://opensensingcity.emse.fr/ontologies/parking/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sosa: <http://www.w3.org/ns/sosa/>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
GENERATE {
  ?parkingIRI a pk:ParkingFacility;
   rdfs:label ?name;
   pk:nbParkingSpaces ?nbParkingSpaces;
    geo:lat ?lat;
    geo:long ?long;
}
SOURCE <http://data.metromobilite.fr/api/bbox/json?types=PAR> AS ?source
```

```
ITERATOR iter:JSONPath(?source,"$.features.*") AS ?parkings
WHERE {
   BIND( IRI(CONCAT("http://opensensingcity.emse.fr/grenoble/parkings/",
       fn:JSONPath(?parkings,"$.properties.CODE"))) AS ?parkingIRI )
    BIND( fn:JSONPath(?parkings,"$.properties.LIBELLE") AS ?name )
   BIND( fn:JSONPath(?parkings,"$.properties.TOTAL") AS ?nbParkingSpaces
       )
  BIND( fn:JSONPath(?parkings,"$.geometry.coordinates[0]") AS ?long )
  BIND( fn:JSONPath(?parkings,"$.geometry.coordinates[1]") AS ?lat )
}
```

Listing D.7: Lifting rule for static data source

```
PREFIX iter: <http://w3id.org/sparql-generate/iter/>
PREFIX fn: <http://w3id.org/sparql-generate/fn/>
PREFIX pk: <http://opensensingcity.emse.fr/ontologies/parking/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sosa: <http://www.w3.org/ns/sosa/>
PREFIX time: <http://www.w3.org/2006/time#>
GENERATE {
  ?parkingIRI a pk:ParkingFacility;
  sosa:observation [ a sosa:Observation;
  time:inTimePosition [
  time:numericPosition ?time;
  ];
  pk:nbAvailableParkingSpaces ?avParkingSpaces;
  ];
}
SOURCE <http://data.metromobilite.fr/api/dyn/PAR/json> AS ?source
ITERATOR iter:JSONListKeys(?source) AS ?key
ITERATOR iter:JSONPath(?source,CONCAT("$.",?key)) as ?elements
ITERATOR iter:JSONPath(?elements,"$.*") as ?element
WHERE {
  BIND( IRI(CONCAT("http://opensensingcity.emse.fr/grenoble/parkings/",?
      key)) AS ?parkingIRI )
  BIND(fn:JSONPath(?element,'$.time') AS ?time )
  BIND(fn:JSONPath(?element,'$.dispo') AS ?parkingSpaces )
  BIND(if(?parkingSpaces = -1,0,?parkingSpaces) as ?avParkingSpaces )
}
```

Listing D.8: Lifting rule for dynamic data source

D.4 Heterogeneous LDP Generation

In *Heterogeneous LDP Generation* (Section 6.2.3), we used a design document that refers to a CSV and JSON data source. The design document is shown in Listing D.9. The lifting rule for the CSV and JSON data source is shown in Listing D.10 and Listing D.11 respectively.

```
@prefix : <http://opensensingcity.emse.fr/LDPDesignVocabulary/> .
@prefix data: <http://opensensingcity.emse.fr/LDPDesign/data/> .
@prefix dcat: <http://www.w3.org/ns/dcat#>
@prefix osc:<http://opensensingcity.emse.fr/ontology/>

data:parisData a :NullContainerMap;
   :iriTemplate "paris";
```

```
    :containerMap data:bicycleStationsCM;
    :containerMap data:carSharingStationsCM;
.

data:bicycleStationsCM a :NullContainerMap;
    :iriTemplate "BicycleStations";
    :nonContainerMap data:bicycleStationNM;
.

data:carSharingStationsCM a :NullContainerMap;
    :iriTemplate "CarSharingStations";
    :nonContainerMap data:carSharingStationNM;
.

data:bicycleStationNM a :NonContainerMap;
    :iriTemplate "$path(res,3)$";
    :resourceMap data:bicycleStationRM;
.

data:carSharingStationNM a :NonContainerMap;
    :iriTemplate "$path(res,3)$";
    :resourceMap data:carSharingStationRM;
.

data:bicycleStationRM a :ResourceMap;
    :resourceQuery "{ ?res a osc:BicycleStation .}";
    :graphQuery "CONSTRUCT { ?res ?p ?o . } WHERE { ?res ?p ?o . }";
    :dataSource data:DataSource1 .

data:carSharingStationRM a :ResourceMap;
    :resourceQuery "{ ?res a osc:CarSharingFacility .}";
    :graphQuery "CONSTRUCT { ?res ?p ?o . } WHERE { ?res ?p ?o . }";
    :dataSource data:DataSource2 .

data:DataSource1 a :RDFSource;
    :location "https://opendata.paris.fr/explore/dataset/velib-emplacement-
        des-stations/download/?format=geojson&timezone=Europe/Berlin";
    :liftingRule "https://raw.githubusercontent.com/noorbakerally/
        ParisDataPlatform/master/velib-emplacement-des-stations.json.rqg";
.

data:DataSource2 a :RDFSource;
    :location "https://opendata.paris.fr/explore/dataset/liste-des-
        stations-de-services-de-vehicules/download/?format=csv&timezone=
        Europe/Berlin&use_labels_for_header=true";
    :liftingRule "https://raw.githubusercontent.com/noorbakerally/
        ParisDataPlatform/master/liste-des-stations-de-services-de-
        vehicules.csv.rqg";
.
```

Listing D.9: Design document used in *Heterogeneous LDP Generation* (Section 6.2.3)

```
PREFIX iter: <http://w3id.org/sparql-generate/iter/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX fn: <http://w3id.org/sparql-generate/fn/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX osc:<http://opensensingcity.emse.fr/city/paris/car_sharing_station
    />
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mbv:<http://schema.mobivoc.org/#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
```

```
GENERATE {
  ?stationURI geo:lat ?lat;
     geo:long ?long; .
}
SOURCE <http://example.org/document#0> as ?message
ITERATOR iter:CSV(?message) AS ?station
WHERE {
   BIND( fn:CSV(?station, "CODE_POST" ) AS ?id )
   BIND (URI(CONCAT("http://opensensingcity.emse.fr/city/paris/
       car_sharing_station/",?id)) AS ?stationURI)
   BIND( xsd:double(fn:CSV(?station, "lat" )) AS ?lat )
  BIND( xsd:double(replace(fn:CSV(?station, "long" )," ","")) AS ?long )
}
```

Listing D.10: Lifting Rule in SPARQL-Generate for CSV data source

```
PREFIX iter: <http://w3id.org/sparql-generate/iter/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX fn: <http://w3id.org/sparql-generate/fn/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX osc:<http://opensensingcity.emse.fr/city/paris/station/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mbv:<http://schema.mobivoc.org/#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
GENERATE {
  ?stationURI rdfs:label ?name;
       mbv:totalCapacity ?capacity;
       geo:lat ?lat;
       geo:long ?long;

    .
}
SOURCE <http://example.org/document#0> as ?message
ITERATOR iter:JSONPath(?message,"$.features.*") AS ?station
WHERE {
    BIND( fn:JSONPath(?station, "$.properties.station_id" ) AS ?id )
    BIND( fn:JSONPath(?station, "$.properties.name" ) AS ?name )
    BIND (URI(CONCAT("http://opensensingcity.emse.fr/city/paris/station/",
        str(?id))) AS ?stationURI)
    BIND( xsd:integer(fn:JSONPath(?station, "$.properties.capacity" )) AS
        ?capacity )
    BIND( xsd:double(fn:JSONPath(?station, "$.properties.lat" )) AS ?lat )
    BIND( xsd:double(fn:JSONPath(?station, "$.properties.lon" )) AS ?long
        )
}
```

Listing D.11: Lifting Rule in SPARQL-Generate for JSON data source

D.5 Flexible Design

In Flexible Design, we demonstrate the flexibility of LDP-DL by using several design documents. Here we provide these design documents in Appendix D.5.1 and then in Appendix D.5.2, we describe how to access the LDPs generated using these design documents.

D.5.1 Design Documents

In Chapter 6 (Section 6.2.1), we already provided the first design document. Here, we provide the remaining four below.

Second Design Document

```
@prefix : <https://w3id.org/ldpdl/#> .
@prefix data: <http://opensensingcity.emse.fr/LDPDesign/data/> .
@prefix dcat: <http://www.w3.org/ns/dcat#>

data:catalogCM a :ContainerMap;
   :slugTemplate "$path(?{res},2)$";
   :resourceMap data:catalogRM;
   :containerMap data:datasetsCM .

data:datasetsCM a :NullContainerMap;
   :containerMap data:datasetCM;
   :slugTemplate "datasets" .

data:datasetCM a :ContainerMap;
   :slugTemplate "$path(?{res},4)$";
   :containerMap data:distributionsCM;
   :resourceMap data:datasetRM .

data:distributionsCM a :NullContainerMap;
   :nonContainerMap data:distributionNM;
   :slugTemplate "distributions" .

data:distributionNM a :NonContainerMap;
   :slugTemplate "$path(?{res},4)$";
   :resourceMap data:distributionRM .

data:distributionRM a :ResourceMap;
   :resourceQuery "{ ?{parent.parent} dcat:distribution ?{res} .}";
   :graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o . }";
   :dataSource data:DataSource1 .

data:catalogRM a :ResourceMap;
   :resourceQuery "{ ?{res} a dcat:Catalog .}";
   :graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o .
       FILTER (?p not in (dcat:dataset)) }";
   :dataSource data:DataSource1 .

data:datasetRM a :ResourceMap;
   :resourceQuery "{ ?{parent.parent} dcat:dataset ?{res} .}";
   :graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o .
       FILTER (?p not in (dcat:distribution)) }";
   :dataSource data:DataSource1 .
```

Listing D.12: Second design document used in Experiment 1

Third Design Document

```
@prefix : <https://w3id.org/ldpdl/#> .
@prefix data: <http://opensensingcity.emse.fr/LDPDesign/data/> .
```

```
@prefix dcat: <http://www.w3.org/ns/dcat#>

data:catalogCM a :ContainerMap;
   :slugTemplate "$path(?{res},2)$";
   :resourceMap data:catalogRM;
   :containerMap data:datasetsCM .

data:datasetsCM a :NullContainerMap;
   :containerMap data:datasetCM;
   :slugTemplate "datasets" .

data:datasetCM a :ContainerMap;
   :slugTemplate "$path(?{res},4)$";
   :containerMap data:distributionsCM;
   :resourceMap data:datasetRM .

data:distributionsCM a :NullContainerMap;
   :nonContainerMap data:distributionNM;
   :slugTemplate "distributions" .

data:distributionNM a :NonContainerMap;
   :slugTemplate "$path(?{res},4)$";
   :resourceMap data:distributionRM .

data:distributionRM a :ResourceMap;
   :resourceQuery "{ ?{parent.parent} dcat:distribution ?{res} .}";
   :graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o . }";
   :dataSource data:DataSource1 .

data:catalogRM a :ResourceMap;
   :resourceQuery "{ ?{res} a dcat:Catalog .}";
   :graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o .
       FILTER (?p not in (dcat:dataset)) }";
   :dataSource data:DataSource1 .

data:datasetRM a :ResourceMap;
   :resourceQuery "{ ?{parent.parent} dcat:dataset ?{res} .}";
   :graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o .
       FILTER (?p not in (dcat:distribution)) }";
   :dataSource data:DataSource1 .
```

Listing D.13: Third design document used in Experiment 1

Fourth Design Document

```
@prefix : <https://w3id.org/ldpdl/#> .
@prefix data: <http://opensensingcity.emse.fr/LDPDesign/data/> .
@prefix dcat: <http://www.w3.org/ns/dcat#>

data:catalogCM a :ContainerMap;
   :slugTemplate "$path(?{res},2)$";
   :resourceMap data:catalogRM;
   :containerMap data:datasetsCM .

data:datasetsCM a :NullContainerMap;
   :containerMap data:datasetCM;
   :slugTemplate "datasets" .

```

```
data:datasetCM a :ContainerMap;
   :slugTemplate "$path(?{res},4)$";
   :containerMap data:distributionsCM;
   :resourceMap data:datasetRM .

data:distributionsCM a :NullContainerMap;
   :nonContainerMap data:distributionNM;
   :slugTemplate "distributions" .

data:distributionNM a :NonContainerMap;
   :slugTemplate "$path(?{res},4)$";
   :resourceMap data:distributionRM .

data:distributionRM a :ResourceMap;
   :resourceQuery "{ ?{parent.parent} dcat:distribution ?{res} .}";
   :graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o . }";
   :dataSource data:DataSource1 .

data:catalogRM a :ResourceMap;
   :resourceQuery "{ ?{res} a dcat:Catalog .}";
   :graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o .
      FILTER (?p not in (dcat:dataset)) }";
   :dataSource data:DataSource1 .

data:datasetRM a :ResourceMap;
   :resourceQuery "{ ?{parent.parent} dcat:dataset ?{res} .}";
   :graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o .
      FILTER (?p not in (dcat:distribution)) }";
   :dataSource data:DataSource1 .
```

Listing D.14: Fourth design document used in Experiment 1

Fifth Design Document

```
@prefix : <https://w3id.org/ldpdl/#> .
@prefix data: <http://opensensingcity.emse.fr/LDPDesign/data/> .
@prefix dcat: <http://www.w3.org/ns/dcat#>

data:catalogCM a :ContainerMap;
   :slugTemplate "$path(?{res},2)$";
   :resourceMap data:catalogRM;
   :containerMap data:datasetsCM .

data:datasetsCM a :NullContainerMap;
   :containerMap data:datasetCM;
   :slugTemplate "datasets" .

data:datasetCM a :ContainerMap;
   :slugTemplate "$path(?{res},4)$";
   :containerMap data:distributionsCM;
   :resourceMap data:datasetRM .

data:distributionsCM a :NullContainerMap;
   :nonContainerMap data:distributionNM;
   :slugTemplate "distributions" .

data:distributionNM a :NonContainerMap;
   :slugTemplate "$path(?{res},4)$";
```

```
25    :resourceMap data:distributionRM .
26
27 data:distributionRM a :ResourceMap;
28    :resourceQuery "{ ?{parent.parent} dcat:distribution ?{res} .}";
29    :graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o . }";
30    :dataSource data:DataSource1 .
31
32 data:catalogRM a :ResourceMap;
33    :resourceQuery "{ ?{res} a dcat:Catalog .}";
34    :graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o .
          FILTER (?p not in (dcat:dataset)) }";
35    :dataSource data:DataSource1 .
36
37 data:datasetRM a :ResourceMap;
38    :resourceQuery "{ ?{parent.parent} dcat:dataset ?{res} .}";
39    :graphQuery "CONSTRUCT { ?{res} ?p ?o . } WHERE { ?{res} ?p ?o .
          FILTER (?p not in (dcat:distribution)) }";
40    :dataSource data:DataSource1 .
```

Listing D.15: Fifth design document used in Experiment 1

D.5.2 Generated LDPs

We use the above five design documents with respect to 21 data sources obtained from different data portals. Below, we provide the URL of these data portals and the LDPs generated to expose their DCAT catalog based on the first design document. The LDPs generated for the remaining four design documents can be obtained by changing the text in the URL of the LDP. For example, the LDP per the second design document for open data angers is `http://opensensingcity.emse.fr/ldpdfend/angers/d2/catalog`. The only difference with the URL of the LDP based on first design as shown below is the 'd2' that refers to the second design document. Likewise, we can change it 'd3', 'd4' and 'd5' for the third, fourth and fifth design document.

Open Data Angers

- URL of Data Portal: https://data.angers.fr/explore/?sort=modified
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/angers/d1/catalog

DataTourism62

- URL of Data Portal: https://tourisme62.opendatasoft.com
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/tourism62/d1/catalog

Bistrotdepays

- URL of Data Portal: https://bistrotdepays.opendatasoft.com
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/bistrotdepays/d1/catalog

dataNova

- URL of Data Portal: https://datanova.legroupe.laposte.fr
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/datanova/d1/catalog

Data Sarthe

- URL of Data Portal: https://data.sarthe.fr
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/sarthe/d1/catalog

Data Enedis

- URL of Data Portal: https://data.enedis.fr
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/enedis/d1/catalog

Open data hauts-de-seine

- URL of Data Portal: https://opendata.hauts-de-seine.fr
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/hauts-de-seine/d1/catalog

Grand Poitiers Open Data

- URL of Data Portal: https://data.grandpoitiers.fr
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/grandpoitiers/d1/catalog

Data Ile de France

- URL of Data Portal: https://data.iledefrance.fr/
- URL of LDP:http://opensensingcity.emse.fr/ldpdfend/iledefrance/d1/catalog

Data Info Locale

- URL of Data Portal: https://datainfolocale.opendatasoft.com/
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/datainfolocale/d1/catalog

Open Data La Haute-garonne

- URL of Data Portal: https://data.haute-garonne.fr
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/haute-garonne/d1/catalog

Navitia

- URL of Data Portal: https://navitia.opendatasoft.com
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/navitia/d1/catalog

Open Data Corsia

- URL of Data Portal: https://www.data.corsica
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/corsica/d1/catalog

Paris Data

- URL of Data Portal: https://opendata.paris.fr
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/paris/d1/catalog

Data Ratp

- URL of Data Portal: https://data.ratp.fr
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/ratp/d1/catalog

Rennes Metropole

- URL of Data Portal: https://data.iledefrance.fr/
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/rennesmetropole/d1/catalog

SNCF Open Data

- URL of Data Portal: https://data.iledefrance.fr/
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/sncf/d1/catalog

Ile de France Mobilite

- URL of Data Portal: https://opendata.stif.info
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/stif/d1/catalog

Data Toulouse Metropole

- URL of Data Portal: https://data.toulouse-metropole.fr
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/toulouse/d1/catalog

Ville D'Agen Open Data

- URL of Data Portal: https://data.agen.fr
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/agen/d1/catalog

DATA ISSY.com

- URL of Data Portal: https://data.issy.com
- URL of LDP: http://opensensingcity.emse.fr/ldpdfend/issy/d1/catalog

Bibliography

[ABH15] Sandro Hawke Alexandre Bertails and Ivan Herman. Linked Data Platform (LDP) Working Group Charter. Technical report, World Wide Web Consortium (W3C), July 31 2015.

[ABPS12] Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux, and Juan Sequeda. A Direct Mapping of Relational Data to RDF, W3C Recommendation 27 September 2012. W3C Recommendation, World Wide Web Consortium (W3C), September 27 2012.

[ADL+09] Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumueller. Triplify: light-weight linked data publication from relational databases. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 621–630, 2009.

[AF05] Antonio Amorin and Gary Figgins. Method of conducting data quality analysis, May 19 2005. US Patent App. 10/953,728.

[AHSB12] Ben Adida, Ivan Herman, Manu Sporny, and Mark Birbeck. RDFa 1.1 Primer - Rich Structured Data Markup for Web Documents, W3C Working Group Note 07 June 2012. W3C Note, World Wide Web Consortium (W3C), June 7 2012.

[AHUV13] Carlos Buil Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. SPARQL web-querying infrastructure: Ready for action? In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II*, pages 277–293, 2013.

[AKKP08a] W. Akhtar, J. Kopeckỳ, T. Krennwallner, and A. Polleres. XSPARQL: Traveling between the XML and RDF worlds–and avoiding the XSLT pilgrimage. In *ESWC*, 2008.

[AKKP08b] Waseem Akhtar, Jacek Kopecký, Thomas Krennwallner, and Axel Polleres. XSPARQL: traveling between the XML and RDF worlds - and avoiding the XSLT pilgrimage. In *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008, Proceedings*, pages 432–447, 2008.

[All10] Subbu Allamaraju. *RESTful Web Services Cookbook*. O'Reilly, 2010.

[AN15] S. Abdellaoui and F. Nader. Semantic data warehouse at the heart of competitive intelligence systems: Design approach. In *2015 6th International Conference on Information Systems and Economic Intelligence (SIIE)*, pages 141–145, Feb 2015.

[BBL08] David Beckett and Tim Berners-Lee. Turtle - Terse RDF Triple Language, W3C Team Submission 14 January 2008. W3C team submission, World Wide Web Consortium (W3C), January 14 2008.

[BBZ16] Noorani Bakerally, Olivier Boissier, and Antoine Zimmermann. Smart city artifacts web portal. In *The Semantic Web - ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 - June 2, 2016, Revised Selected Papers*, pages 172–177, 2016.

[BC06] Christian Bizer and Richard Cyganiak. D2R server-publishing relational databases on the semantic web. In *Poster at the 5th ISWC*, volume 175, 2006.

[BC15] Sebastian K. Boell and Dubravka Cecez-Kecmanovic. What is an information system? In *48th Hawaii International Conference on System Sciences, HICSS 2015, Kauai, Hawaii, USA, January 5-8, 2015*, pages 4959–4968, 2015.

[BCW12] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182, 2012.

[BG14] Dan Brickley and Ramanathan V. Guha. RDF Schema 1.1. W3C Recommendation, World Wide Web Consortium (W3C), February 25 2014.

[BHB09] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[Bis98] Yaser A. Bishr. Overcoming the semantic and other barriers to GIS interoperability. *International Journal of Geographical Information Science*, 12(4):299–314, 1998.

[BKLW99] Susanne Busse, Ralf-Detlef Kutsche, Ulf Leser, and Herbert Weber. Federated information systems: Concepts, terminology and architectures. 1999.

[BL06] Tim Berners-Lee. *Linked Data-Design Issues*, 2006.

[Bro10] Michael L. Brodie. Data integration at scale: From relational data integration to information ecosystems. In *24th IEEE International Conference on Advanced Information Networking and Applications, AINA 2010, Perth, Australia, 20-13 April 2010*, pages 2–3, 2010.

[BS04] Christian Bizer and Andy Seaborne. D2rq-treating non-rdf databases as virtual rdf graphs. In *Proceedings of the 3rd international semantic*

web conference (ISWC2004), volume 2004. Proceedings of ISWC2004, 2004.

[BWF13] Susanne Becker, Volker Walter, and Dieter Fritsch. Modeling concepts for consistency analysis of multiple representations and heterogeneous 3d geodata. In *Information Fusion and Geographic Information Systems, IF&GIS 2013 - Environmental and Urban Challenges, May 12-15, 2013, St. Petersburg, Russia*, pages 91–106, 2013.

[BWLR12] Steve Battle, David Wood, James Leigh, and Luke Ruth. The callimachus project: Rdfa as a web template language. In *Proceedings of the Third International Workshop on Consuming Linked Data, COLD 2012, Boston, MA, USA, November 12, 2012*, 2012.

[BZ17] Mohammad Noorani Bakerally and Antoine Zimmermann. Towards the automatic deployment of data in linked data platforms. In *Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 23rd - to - 25th, 2017.*, 2017.

[Car07] Jorge S. Cardoso. Developing course management systems using the semantic web. In *The Semantic Web: Real-World Applications from Industry*, pages 169–188. 2007.

[CG17] Sarven Capadisli and Amy Guy. Linked Data Notifications. Technical report, World Wide Web Consortium (W3C), May 2 2017.

[CGHP+12] Bernardo Cuenca-Grau, Ian Horrocks, Bijan Parsia, Alan Rutthenberg, and Michael Schneider. OWL 2 Web Ontology Language Mapping to RDF Graphs, W3C Recommendation 11 December 2012. W3C Recommendation, World Wide Web Consortium (W3C), December 10 2012.

[Cha13a] Pierre-Antoine Champin. RDF-REST: A unifying framework for web apis and linked data. In *Proceedings of the First Workshop on Services and Applications over Linked APIs and Data co-located with the 10th Extended Semantic Web Conference (ESWC 2013), Montpellier, France, May 26, 2013.*, pages 10–19, 2013.

[Cha13b] Calvin M. L. Chan. From open data to open innovation strategies: Creating e-services using open government data. In *46th Hawaii International Conference on System Sciences, HICSS 2013, Wailea, HI, USA, January 7-10, 2013*, pages 1890–1899, 2013.

[CHM08] Peter Coetzee, Tom Heath, and Enrico Motta. Sparqplug: Generating linked data from legacy html, SPARQL and the DOM. In *Proceedings of the WWW2008 Workshop on Linked Data on the Web, LDOW 2008, Beijing, China, April 22, 2008.*, 2008.

[CJ06] Bengt Carlsson and Andreas Jacobsson. Security consistency in information ecosystems: structuring the risk environment on the internet. *Journal of Information System Security*, 2(1):3–26, 2006.

[CKRJ17] Javad Chamanara, Birgitta König-Ries, and H. V. Jagadish. Quis: In-situ heterogeneous data source querying. *Proc. VLDB Endow.*, 10(12):1877–1880, August 2017.

[CL17] Simon Cox and Chris Little. Time Ontology in OWL W3C Recommendation 19 October 2017. W3C Candidate Recommendation, World Wide Web Consortium (W3C), October 11 2017.

[Com17] European Commission. New european interoperability framework, 2017.

[Con07] Dan Connolly. Gleaning Resource Descriptions from Dialects of Languages (GRDDL), W3C Recommendation 11 September 2007. W3C Recommendation, World Wide Web Consortium (W3C), September 11 2007.

[CS14a] Gavin Carothers and Andy Seaborne. RDF 1.1 N-Triples - A line-based syntax for RDF graph, W3C Recommendation 25 February 2014. W3C Recommendation, World Wide Web Consortium (W3C), February 25 2014.

[CS14b] Gavin Carothers and Andy Seaborne. RDF 1.1 TriG - RDF Dataset Language, W3C Recommendation 25 February 2014. W3C Recommendation, World Wide Web Consortium (W3C), February 25 2014.

[CWL14] R. Cyganiak, D. Wood, and M. Lanthaler. RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation 25 February 2014. Technical report, W3C, 2014.

[CZ06] Lois Mai Chan and Marcia Lei Zeng. Metadata interoperability and standardization - A study of methodology, part I: achieving interoperability at the schema level. *D-Lib Magazine*, 12(6), 2006.

[CZdS18] Andreiwid Sheffer Corrêa, Pär-Ola Zander, and Flávio Soares Corrêa da Silva. Investigating open data portals automatically: a methodology and some illustrations. In *Proceedings of the 19th Annual International Conference on Digital Government Research: Governance in the Data Age, DG.O 2018, Delft, The Netherlands, May 30 - June 01, 2018*, pages 82:1–82:10, 2018.

[DD99] Ruxandra Domenig and Klaus R. Dittrich. An overview and classification of mediated query systems. *SIGMOD Record*, 28(3):63–72, 1999.

[DD11] Leigh Dodds and Ian Davis. Linked data patterns. *Online: http://patterns. dataincubator. org/book*, 2011.

[DFJ+04] Li Ding, Timothy W. Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: a search and metadata engine for the semantic web. In *Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA, November 8-13, 2004*, pages 652–659, 2004.

[dM11] Mathieu d'Aquin and Enrico Motta. Watson, more than a semantic web search engine. *Semantic Web*, 2(1):55–63, 2011.

[DPLB14] Daniele Dell'Aglio, Axel Polleres, Nuno Lopes, and Stefan Bischof. Querying the web of data with XSPARQL 1.1. In *Proceedings of the 2014 International Conference on Developers-Volume 1268*, pages 113–118. CEUR-WS. org, 2014.

[DS05] Martin J. Dürst and Michel Suignard. Internationalized Resource Identifiers (IRIs). Technical report, Internet Engineering Task Force, January 2005.

[DSC] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language, W3C Recommendation 27 September 2012. Technical report.

[DVSC+14] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle. RML: A generic language for integrated RDF mappings of heterogeneous data. In *LDOW*, 2014.

[DW05a] Elizabeth Daniel and John Ward. Enterprise portals: Addressing the organizational and individual perspectives of information systems. *ECIS 2005 Proceedings*, page 3, 2005.

[DW05b] Elizabeth M. Daniel and Andrew D. White. The future of inter-organisational system linkages: findings of an international delphi study. *EJIS*, 14(2):188–203, 2005.

[EGMGC14] M. Esteban-Gutiérrez, N. Mihindukulasooriya, and R. García-Castro. LDP4j: A framework for the development of interoperable read-write Linked Data applications. In *ISWC Developers Workshop*, 2014.

[FC16] André Freitas and Edward Curry. Big data curation. In *New Horizons for a Data-Driven Economy - A Roadmap for Usage and Exploitation of Big Data in Europe*, pages 87–118. 2016.

[FJ97] Mariano Ferndndez and Natalia Juristo. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. pages 33–40, 1997.

[FKA+12] Philipp Frischmuth, Jakub Klímek, Sören Auer, Sebastian Tramp, and Jörg Unbehauen. Linked data in enterprise information integration. 2012.

[FL11] Rainer Larin Fonseca and Eduardo Garea Llano. Automatic representation of geographical data from a semantic point of view through a new ontology and classification techniques. *Trans. GIS*, 15(1):61–85, 2011.

[FMW+14] Simon Foster, Alvaro Miyazawa, Jim Woodcock, Ana Cavalcanti, John S. Fitzgerald, and Peter Gorm Larsen. An approach for managing semantic heterogeneity in systems of systems engineering. In *9th International Conference on System of Systems Engineering, SoSE 2014, Glenelg, Australia, June 9-13, 2014*, pages 113–118, 2014.

[For06] Forrester. Companies Adopt Employee Portals, Not Portal Best Practices, 2006.

[Fox96] SM Fox. Research in enterprise integration. In *Proceedings Artificial Intelligence and Manufacturing Research Planning Workshop*, pages 40–52, 1996.

[FR07] R. B. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *FOSE*, 2007.

[GA18] Didem GÃijrdÃijr and Fredrik Asplund. A systematic review to merge discourses: Interoperability, integration and cyber-physical systems. *Journal of Industrial Information Integration*, 9:14 – 23, 2018.

[Gag07] Michel Gagnon. Ontology-based integration of data sources. In *10th International Conference on Information Fusion, FUSION 2007, Québec, Canada, July 9-12, 2007*, pages 1–8, 2007.

[Gdh07] J. Gregorio and B. de hOra. The Atom Publishing Protocol. Technical report, IETF, 2007.

[Gia04] Ronald E Giachetti. A framework to review the information integration of the enterprise. *International Journal of Production Research*, 42(6):1147–1166, 2004.

[Goe08] Frank G. Goethals. Important issues for evaluating inter-organizational data integration configurations. 2008.

[GPR+17] Aldo Gangemi, Valentina Presutti, Diego Reforgiato Recupero, Andrea Giovanni Nuzzolese, Francesco Draicchio, and Misael Mongiovì. Semantic web machine reading with FRED. *Semantic Web*, 8(6):873–893, 2017.

[Gro07] Kshitij Grover. Kenneth c. laudon and jane p. laudon, management information system - managing the digital firm (ninth ed.), prentice-hall, new jersey (2005) ISBN: 0-131-53841-1. *Inf. Process. Manage.*, 43(6):1833–1834, 2007.

[Gro13] W3C SPARQL Working Group. SPARQL 1.1 Overview. W3C Recommendation, World Wide Web Consortium (W3C), March 21 2013.

[GS14] Fabien Gandon and Guus Schreiber. RDF 1.1 XML Syntax. W3C Recommendation, World Wide Web Consortium (W3C), February 25 2014.

[GS16] José Ramón Gil-García and Djoko Sigit Sayogo. Government inter-organizational information sharing initiatives: Understanding the main determinants of success. *Government Information Quarterly*, 33(3):572–582, 2016.

[GSB16] Seymour Goodman, Detmar W Straub, and Richard Baskerville. *Information Security: Policy, Processes, and Practices*. Routledge, 2016.

[Hal05] Alon Y. Halevy. Why your data won't mix. *ACM Queue*, 3(8):50–58, 2005.

[Haw13] Sandro Hawke. Sparql query results xml format. W3C Recommendation, World Wide Web Consortium (W3C), March 21 2013.

[Haz02] Tushar K. Hazra. Building enterprise portals: principles to practice. In *Proceedings of the 24th International Conference on Software Engineering, ICSE 2002, 19-25 May 2002, Orlando, Florida, USA*, pages 623–633, 2002.

[HG01] Farshad Hakimpour and Andreas Geppert. Resolving semantic heterogeneity in schema integration. In *FOIS*, pages 297–308, 2001.

[HK18] Hafiz Mahfooz Ul Haque and Sajid Ullah Khan. A context-aware reasoning framework for heterogeneous systems. In *Advancements in Computational Sciences (ICACS), 2018 International Conference on*, pages 1–9. IEEE, 2018.

[HMTF09] N Heydari, A Mansourian, M Taleai, and GR Fallahi. Ontology-based gis web service for increasing semantic interoperability among organizations involving drilling in city of tehran. In *11th GSDI World Conference (GSDI 11), Spatial Data Infrastructure Convergence: Building SDI Bridges to Address Global Challenges, POSTER FORUM, Rotterdam, The Netherlands*, 2009.

[HNS12] Arnaud Le Hors, Martin Nally, and Steve Speicher. Using read/write linked data for application integration - towards a linked data basic profile. In *WWW2012 Workshop on Linked Data on the Web, Lyon, France, 16 April, 2012*, 2012.

[Hog14] Aidan Hogan. Linked data & the semantic web standards. In *Linked Data Management.*, pages 3–48. 2014.

[HS13a] S. Harris and A. Seaborne. SPARQL 1.1 Query Language, W3C Recommendation 21 March 2013. Technical report, W3C, 2013.

[HS13b] Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language, W3C Recommendation 21 March 2013. W3C Recommendation, World Wide Web Consortium (W3C), March 21 2013.

[HTD09] Sally Rao Hill, Indrit Troshani, and Robyn Davidson. Developing an e-collaboration framework for knowledge sharing in the australian wine-making industry: Research in progress. 2009.

[HTP+09] Jeffery S. Horsburgh, David G. Tarboton, Michael Piasecki, David R. Maidment, Ilya Zaslavsky, David Valentine, and Thomas Whitenack. An integrated system for publishing environmental observations data. *Environmental Modelling and Software*, 24(8):879–888, 2009.

[Hug07] Mats-Åke Hugoson. Centralized versus decentralized information systems - A historical flashback. In *History of Nordic Computing 2 - Second IFIP WG 9.7 Conference, HiNC2, Turku, Finland, August 21-23, 2007, Revised Selected Papers*, pages 106–115, 2007.

[HVdB11] Noor Huijboom and Tijs Van den Broek. Open data: an international comparison of strategies. *European journal of ePractice*, 12(1):4–16, 2011.

[IEE91] Ieee standard computer dictionary: A compilation of ieee standard computer glossaries. *IEEE Std 610*, pages 1–217, Jan 1991.

[Jac04] Andreas Jacobsson. *Exploring Privacy Risks in Information Networks*. Citeseer, 2004.

[Jau99] Olivier Jautzy. Interoperable databases: A programming language approach. In *1999 International Database Engineering and Applications Symposium, IDEAS 1999, Montreal, Canada, August 2-4, 1999, Proceedings*, pages 63–71, 1999.

[Joh17] Lisa R Johnston. Summary of the" data curation handbook steps" from curating research data volume two: A handbook of current practice. Association of College & Research Libraries, 2017.

[JSC18] Mohammadreza Jelokhani-Niaraki, Abolghasem Sadeghi-Niaraki, and Soo-Mi Choi. Semantic interoperability of GIS and MCDA tools for environmental assessment and decision making. *Environmental Modelling and Software*, 100:104–122, 2018.

[KA15] K. Kurniawan and A. Ashari. Service orchestration using enterprise service bus for real-time government executive dashboard system. In *2015 International Conference on Data and Software Engineering (ICoDSE)*, pages 207–212, Nov 2015.

[Kin08] R. L. King. Information services for smart grids. In *2008 IEEE Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century*, pages 1–5, July 2008.

[KLM+11] Sotirios Koussouris, Fenareti Lampathaki, Spiros Mouzakitis, Yannis Charalabidis, and John Psarras. Digging into the real-life enterprise interoperability areas definition and overview of the main research areas. *Proceedings of CENT*, pages 19–22, 2011.

[KML14] Sotirios Koussouris, Spiros Mouzakitis, and Fenareti Lampathaki. A taxonomy of scientific areas driving assessment of organisations readiness. In *Revolutionizing Enterprise Interoperability through Scientific Foundations*, pages 24–40. IGI Global, 2014.

[KMMV07] Spyros Kitsiou, Aristides Matopoulos, Vicky Manthou, and Maro Vlachopoulou. Evaluation of integration technology approaches in the healthcare supply chain. *International Journal of Value Chain Management*, 1(4):325–343, 2007.

[KP09] Yannis Katsis and Yannis Papakonstantinou. View-based data integration. In *Encyclopedia of Database Systems*, pages 3332–3339. 2009.

[KRT+16] Sylvain Kubler, Jérémy Robert, Yves Le Traon, Jürgen Umbrich, and Sebastian Neumaier. Open data portal quality comparison using AHP. In *Proceedings of the 17th International Digital Government Research Conference on Digital Government Research, DG.O 2016, Shanghai, China, June 08 - 10, 2016*, pages 397–407, 2016.

[KTM17] Maulik R. Kamdar, Tania Tudorache, and Mark A. Musen. A systematic analysis of term reuse and term overlap across biomedical ontologies. *Semantic Web*, 8(6):853–871, 2017.

[KWB03] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA explained - the Model Driven Architecture: practice and promise*. Addison Wesley object technology series. Addison-Wesley, 2003.

[LBC17] Bernadette Farias Lóscio, Caroline Burle, and Newton Calegari. Data on the web best practices. W3c recommendation, World Wide Web Consortium (W3C), January 31 2017.

[LBDP11] Nuno Lopes, Stefan Bischof, Stefan Decker, and Axel Polleres. On the semantics of heterogeneous querying of relational, XML and RDF data with XSPARQL. In *Proceedings of the 15th Portuguese Conference on Artificial Intelligence (EPIA 2011), Lisbon, Portugal*, 2011.

[Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 233–246, 2002.

[Lie16] Jay Liebowitz. *Successes and Failures of Knowledge Management*. Morgan Kaufmann, 2016.

[LIG+16] G. Loseto, S. Ieva, F. Gramegna, M. Ruta, F. Scioscia, and E. Sciascio. Linking the Web of Things: LDP-CoAP Mapping. In *ANT/SEIT Workshops*, 2016.

[LKA+12] Fenareti Lampathaki, Sotirios Koussouris, Carlos Agostinho, Ricardo Jardim-Gonçalves, Yannis Charalabidis, and John E. Psarras. Infusing scientific foundations into enterprise interoperability. *Computers in Industry*, 63(8):858–866, 2012.

[LKdL15] Harshana Liyanage, Paul Krause, and Simon de Lusignan. Using ontologies to improve semantic interoperability in health data. *Journal of innovation in health informatics*, 22(2):309–315, 2015.

[LSH03] Jinyoul Lee, Keng Siau, and Soongoo Hong. Enterprise integration with ERP and EAI. *Commun. ACM*, 46(2):54–60, 2003.

[LSM12] Timothy Lebo, Satya Sahoo, and Deborah L. McGuinness. PROV-O: The PROV Ontology, W3C Candidate Recommendation 11 December 2012. W3C Candidate Recommendation, World Wide Web Consortium (W3C), December 11 2012.

[LZB17a] M. Lefrançois, A. Zimmermann, and N. Bakerally. A SPARQL extension for generating RDF from heterogeneous formats. In *ESWC*, 2017.

[LZB17b] Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. A SPARQL extension for generating RDF from heterogeneous formats. In *The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 - June 1, 2017, Proceedings, Part I*, pages 35–50, 2017.

[Mar12] Tim Martin. Uk location programme view service operational guide. Technical report, Data.gov.uk, 2012. https://data.gov.uk/sites/default/files/View-Service-Operational-Guide-v2_2_10.pdf.

[MB09] Alistair Miles and Sean Bechhofer. SKOS Simple Knowledge Organization System, Reference, W3C Recommendation 18 August 2009. W3C Recommendation, World Wide Web Consortium (W3C), August 18 2009.

[MBB+03] Brahim Medjahed, Boualem Benatallah, Athman Bouguettaya, Anne H. H. Ngu, and Ahmed K. Elmagarmid. Business-to-business interactions: issues and enabling technologies. *VLDB J.*, 12(1):59–85, 2003.

[MBWdlI15] Aalaa Mojahed, Joao H. Bettencourt-Silva, Wenjia Wang, and Beatriz de la Iglesia. Applying clustering analysis to heterogeneous data using similarity matrix fusion (SMF). In *Machine Learning and Data Mining in Pattern Recognition - 11th International Conference, MLDM 2015, Hamburg, Germany, July 20-21, 2015, Proceedings*, pages 251–265, 2015.

[ME14a] F. Maali and J. Erickson. Data Catalog Vocabulary (DCAT), W3C Recommendation 16 January 2014. Technical report, W3C, 2014.

[ME14b] Fadi Maali and John Erickson. Data Catalog Vocabulary (DCAT), W3C Recommendation 16 January 2014. W3C Recommendation, World Wide Web Consortium (W3C), January 16 2014.

[MGG13] N. Mihindukulasooriya, R. Garcia-Castro, and M. E. Gutiérrez. Linked data platform as a novel approach for enterprise application integration. In *COLD*, 2013.

[MGG14] N. Mihindukulasooriya, M. E. Gutiérrez, and R. García-Castro. A linked data platform adapter for the bugzilla issue tracker. In *ISWC Posters & Demo*, pages 89–92, 2014.

[MO04] Andreia Malucelli and Eugénio Oliveira. Ontology-services agent to help in the structural and semantic heterogeneity. In *Working Conference on Virtual Enterprises*, pages 175–182. Springer, 2004.

[MPC+07] Arturo Molina, Hervé Panetto, David Chen, Larry Whitman, Vincent Chapurlat, and Francois B. Vernadat. Enterprise Integration and Networking: challenges and trends. *Studies in Informatics and Control*, 16(4):353–368, December 2007.

[MPC+14] N. Mihindukulasooriya, F. Priyatna, Ó. Corcho, R. García-Castro, and M. E. Gutiérrez. morph-LDP: An R2RML-Based Linked Data Platform Implementation. In *ESWC Poster & Demo*, 2014.

[MPS10] Ming Mao, Yefei Peng, and Michael Spring. An adaptive ontology mapping approach with neural network based constraint satisfaction. *J. Web Sem.*, 8(1):14–25, 2010.

[MPSP12a] Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). W3C Recommendation, World Wide Web Consortium (W3C), December 11 2012.

[MPSP12b] Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 Web Ontology Language XML Serialization (Second Edition). W3C Recommendation, World Wide Web Consortium (W3C), December 11 2012.

[MRW10] Hossein Mohammadi, Abbas Rajabifard, and Ian P. Williamson. Development of an interoperable tool to facilitate spatial data integration in the context of sdi. *International Journal of Geographical Information Science*, 24(4):487–505, 2010.

[MSMV18] Thomas Minier, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. Ulysses: An intelligent client for replicated triple pattern fragments. In *The Semantic Web: ESWC 2018 Satellite Events - ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers*, pages 182–186, 2018.

[MZG14] Ahmad Mehrbod, Aneesh Zutshi, and António Grilo. Semantic and syntactic matching of e-catalogues - using vector space model. In *ICE-B 2014 - Proceedings of the 11th International Conference on e-Business, Vienna, Austria, 28-30 August, 2014*, pages 224–229, 2014.

[NJN] Turkka Näppilä, Kalervo Järvelin, and Timo Niemi. A tool for data cube construction from structurally heterogeneous xml documents. *Journal of the American Society for Information Science and Technology*, 59(3):435–449.

[NPF+13] Trinh Hoang Nguyen, Andreas Prinz, Trond FriisÃÿ, Rolf Nossum, and Ilya Tyapin. A framework for data integration of offshore wind farms. *Renewable Energy*, 60:150 – 161, 2013.

[NS14] Kiyoshi Nitta and Iztok Savnik. Survey of rdf storage managers. In *Proceedings of the 6th international conference on advances in databases, knowledge, and data applications (DBKDAâĂŹ14), Chamonix, France*, pages 148–153, 2014.

[NVS+06] Meenakshi Nagarajan, Kunal Verma, Amit P. Sheth, John A. Miller, and Jon Lathem. Semantic interoperability of web services - challenges and experiences. In *2006 IEEE International Conference on Web Services (ICWS 2006), 18-22 September 2006, Chicago, Illinois, USA*, pages 373–382, 2006.

[Ogb13] Chimezie Ogbuji. SPARQL 1.1 Graph Store HTTP Protocol. W3C Recommendation, World Wide Web Consortium (W3C), March 21 2013.

[Oli17] Antoni Olivé. The universal ontology: A vision for conceptual modeling and the semantic web (invited paper). In *Conceptual Modeling - 36th International Conference, ER 2017, Valencia, Spain, November 6-9, 2017, Proceedings*, pages 1–17, 2017.

[PAG09] Jorge Pérez, Marcelo Arenas, and Claudio Gutiérrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, 2009.

[Pau07] Ray J. Paul. Challenges to information systems: time to change. *EJIS*, 16(3):193–195, 2007.

[PC14] Eric Prud'hommeaux and Gavin Carothers. RDF 1.1 Turtle - Terse RDF Triple Language, W3C Recommendation 25 February 2014. W3C Recommendation, World Wide Web Consortium (W3C), February 25 2014.

[PCDT05] Peter Plessers, Sven Casteleyn, and Olga De Troyer. Semantic web development with wsdm. In *Proceedings 5th International Workshop on Knowledge Markup and Semantic Annotation (SemAnnot 2005)*, pages 1–12, 2005.

[PST04] H Sofia Pinto, Steffen Staab, and Christoph Tempich. Diligent: Towards a fine-grained methodology for distributed, loosely-controlled and evolving. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004)*, volume 110, page 393, 2004.

[PW16] Hervé Panetto and Lawrence E. Whitman. Knowledge engineering for enterprise integration, interoperability and networking: Theory and applications. *Data Knowl. Eng.*, 105:1–4, 2016.

[PZLN18] Zhengyu Pan, Tao Zhu, Hong Liu, and Huansheng Ning. A survey of rdf management technologies and benchmark datasets. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–12, 2018.

[QLS+11] Weijun Qin, Qiang Li, Limin Sun, Hongsong Zhu, and Yan Liu. Restthing: A restful web service infrastructure for mash-up physical and web resources. In *IEEE/IFIP 9th International Conference on Embedded and Ubiquitous Computing, EUC 2011, Melbourne, Australia, October 24-26, 2011*, pages 197–204, 2011.

[RCL14] Reza Rezaei, Thiam Kian Chiew, and Sai Peck Lee. A review on e-business interoperability frameworks. *Journal of Systems and Software*, 93:199–216, 2014.

[RH13] Laurens Rietveld and Rinke Hoekstra. YASGUI: not just another SPARQL client. In *The Semantic Web: ESWC 2013 Satellite Events - ESWC 2013 Satellite Events, Montpellier, France, May 26-30, 2013, Revised Selected Papers*, pages 78–86, 2013.

[RM13] Rajesh Rajaguru and Margaret Jekanyika Matanda. Effects of inter-organizational compatibility on supply chain capabilities: Exploring the mediating role of inter-organizational information systems (iois) integration. *Industrial Marketing Management*, 42(4):620 – 632, 2013. Special Issue on Applied Intelligent Systems in Business-to-Business Marketing.

[RMB+12] Rudolf Reinhard, Tobias Meisen, Thomas Beer, Daniel Schilberg, and Sabina Jeschke. A framework enabling data integration for virtual production. In *Enabling Manufacturing Competitiveness and Economic Sustainability*, pages 275–280. Springer, 2012.

[RNC+03] Vijayshankar Raman, Inderpal Narang, Chris Crone, Laura Haas, Susan Malaika, Tina Mukai, Dan Wolfson, and Chaitan Baru. Services for data access and data processing on grids. *Global Grid Formum Document GFD-I*, 14, 2003.

[RS08] Alistair Russell and Paul R. Smart. NITELIGHT: A graphical editor for SPARQL queries. In *Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC2008), Karlsruhe, Germany, October 28, 2008*, 2008.

[RV16] David Romero and François B. Vernadat. Enterprise information systems state of the art: Past, present and future trends. *Computers in Industry*, 79:3–13, 2016.

[SAD+15] Mohamed-Foued Sriti, Ibrahim Assouroko, Guillaume Ducellier, Philippe Boutinaud, and Benoît Eynard. Ontology–based approach for product information exchange. *International Journal of Product Lifecycle Management*, 8(1):1–23, 2015.

[SAM15a] S. Speicher, J. Arwe, and A. Malhotra. Linked Data Platform 1.0. Technical report, W3C, February 26 2015.

[SAM15b] S. Speicher, J. Arwe, and A. Malhotra. Linked Data Platform Paging 1.0 W3C Working Group Note 30 June 2015. Technical report, W3C, 2015.

[SAM15c] Steve Speicher, John Arwe, and Ashok Malhotra. Linked Data Platform 1.0. Technical report, World Wide Web Consortium (W3C), February 26 2015.

[SBK+12] Sebastian Schaffert, Christoph Bauer, Thomas Kurz, Fabian Dorschel, Dietmar Glachs, and Manuel Fernandez. The linked media framework: integrating and interlinking enterprise media content and data. In *I-SEMANTICS 2012 - 8th International Conference on Semantic Systems, I-SEMANTICS '12, Graz, Austria, September 5-7, 2012*, pages 25–32, 2012.

[Sea13a] Andy Seaborne. Sparql 1.1 query results csv and tsv formats. W3C Recommendation, World Wide Web Consortium (W3C), March 21 2013.

[Sea13b] Andy Seaborne. Sparql 1.1 query results json format. W3C Recommendation, World Wide Web Consortium (W3C), March 21 2013.

[She01] Amit P. Sheth. Changing focus on interoperability in information systems: from system, syntax, structure to semantics. 2001.

[Ska17] Hala Skaf-Molli. *Decentralized Data Management for the Semantic Web . (Gestion décentralisée de données du web sémantique).* 2017.

[SKL14] Manu Sporny, Greg Kellogg, and Markus Lanthaler. JSON-LD 1.0 - A JSON-based Serialization for Linked Data, W3C Recommendation 16 January 2014. W3C Recommendation, World Wide Web Consortium (W3C), February 25 2014.

[SM14] Brian Stein and Alan Morrison. The enterprise data lake: Better integration and deeper analytics. *PwC Technology Forecast: Rethinking integration*, 1:1–9, 2014.

[SR96] Len Seligman and Arnon Rosenthal. A metadata resource to promote data integration. In *IN PROC. OF IEEE METADATA CONFERENCE, SILVER SPRING, MD*, 1996.

[SS06] Steffen Staab and Heiner Stuckenschmidt, editors. *Semantic Web and Peer-to-Peer - Decentralized Management and Exchange of Knowledge and Information.* Springer, 2006.

[SVB+06] T. Stahl, M. Volter, J. Bettin, A. Haase, and S. Helsen. *Model-driven software development: technology, engineering, management.* Pitman, 2006.

[SVVB12] Thanos G. Stavropoulos, Dimitris Vrakas, Danai Vlachava, and Nick Bassiliades. Bonsai: a smart building ontology for ambient intelligence. In *2nd International Conference on Web Intelligence, Mining and Semantics, WIMS '12, Craiova, Romania, June 6-8, 2012*, pages 30:1–30:12, 2012.

[TVN08] Tania Tudorache, Jennifer Vendetti, and Natalya Fridman Noy. Web-protege: A lightweight OWL ontology editor for the web. In *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with the 7th International Semantic Web Conference (ISWC-2008), Karlsruhe, Germany, October 26-27, 2008*, 2008.

[UNP15] Jürgen Umbrich, Sebastian Neumaier, and Axel Polleres. Towards assessing the quality evolution of open data portals. In *Proceedings of ODQ2015: Open Data Quality: from Theory to Practice Workshop, Munich, Germany*, 2015.

[USR10] Nils Urbach, Stefan Smolnik, and Gerold Riempp. An empirical investigation of employee portal success. *J. Strategic Inf. Sys.*, 19(3):184–206, 2010.

[VHM+14] Ruben Verborgh, Olaf Hartig, Ben De Meester, Gerald Haesendonck, Laurens De Vocht, Miel Vander Sande, Richard Cyganiak, Pieter Colpaert, Erik Mannens, and Rik Van de Walle. Querying datasets on the web with high availability. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, pages 180–196, 2014.

[VSH+16] Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. Triple pattern fragments: A low-cost knowledge graph interface for the web. *J. Web Sem.*, 37-38:184–206, 2016.

[VSS02] Ubbo Visser, Heiner Stuckenschmidt, and Christoph Schlieder. Interoperability in gis-enabling technologies. In *Proceedings of the 5th AGILE Conference on Geographic Information Science*, page 291. Citeseer, 2002.

[W3C12] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (Second Edition), W3C Recommendation 11 December 2012. W3C Recommendation, World Wide Web Consortium (W3C), December 11 2012.

[Whi00] Martin White. Enterprise information portals. *The Electronic Library*, 18(5):354–362, 2000.

[Wie92] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.

[Wil07] Elizabeth A Williamson. An evaluation of inter-organisational information systems development on business partnership relations.

International Journal of Business Science and Applied Management, 2(3):36–50, 2007.

[WJ06] Merrill Warkentin and Allen C Johnston. It security governance and centralized security controls. *Enterprise Information Assurance and System Security: Managerial and Technical Issues*, pages 16–24, 2006.

[WJ08] Merrill Warkentin and Allen C Johnston. It governance and organizational design for security management. *Information security: Policies, processes, and practices*, pages 46–68, 2008.

[Wu01] Dong-Jun Wu. Software agents for knowledge management: coordination in multi-agent supply chains and auctions. *Expert Syst. Appl.*, 20(1):51–64, 2001.

[XL02] Zhu Xu and YC Lee. Semantic heterogeneity of geodata. *International archives of photogrammetry remote sensing and spatial information sciences*, 34(4):216–224, 2002.

[Yak07] Elizabeth Yakel. Digital curation. *OCLC Systems & Services*, 23(4):335–340, 2007.

[ZD04] Patrick Ziegler and Klaus R. Dittrich. Three decades of data integration - all problems solved? In *Building the Information Society, IFIP 18th World Computer Congress, Topical Sessions, 22-27 August 2004, Toulouse, France*, pages 3–12, 2004.

École Nationale Supérieure des Mines
de Saint-Étienne

NNT:2018LYSEM029
Mohammad Noorani BAKERALLY
Generation of Linked Data Platforms in Highly Decentralized Information Ecosystem
Speciality: Computer Science
Keywords: Écosystème d'information, Système d'Information, Web sémantique, *Linked Data Platform 1.0*

Abstract:
Information ecosystem with decentralized architectures have known some success in terms of data interoperability. However, today, information ecosystems in which organizations are operating are moving towards highly decentralized architectures for reasons such as globalization, multinationalism or complex supply chain collaborations. The problem in highly decentralized information ecosystem is that while data is heterogeneous, there is little to no coordination between its different units. Consequently, resolving data heterogeneity and enhancing interoperability between the information systems become more challenging in such a context.

In this work, our hypothesis is that Semantic Web technologies and in particular the Linked Data Platform 1.0 (LDP) standard can be used to provide a homogeneous view and access to self-described data. Thus, it decreases the need for contacting data providers to understand and make use of the data. However, while several LDP implementations exist, they provide no support for automating the generation of LDPs. Consequently, deploying LDPs from existing data sources still involves a lot of manual development. To solve this problem, we propose an approach that uses a language to describe how existing data sources can be used to generate LDPs compatible with any implementation of the LDP standard and deployable on any of them. We formally describe the syntax and semantics of language. We provide an implementation of the approach that instantiates an automatized generation and deployment workflow. Finally, we evaluate our language and approach in general by performing several experiments to show how our approach automatizes the generation of LDPs, while enhancing design reusability, from existing data sources that are either heterogeneous or have hosting constraints.

École Nationale Supérieure des Mines
de Saint-Étienne

NNT:2018LYSEM029
Mohammad Noorani BAKERALLY
Generation of Linked Data Platforms in Highly Decentralized Information Ecosystem
Spécialité: Computer Science
Mots clefs: Information Ecosystem, Information System, Semantic Web, Linked Data Platform 1.0

Résumé:
Les écosystèmes d'information avec une architecture décentralisée ont connu des succès dans leur capacité à traiter l'interopérabilité. Cependant aujourd'hui, la gouvernance de ces écosystèmes d'information se transforme et donne lieu à des systèmes fortement décentralisés du fait de leur globalisation et mondialisation, de leur inscription dans des collaborations complexes avec d'autres organisations.

Le problème dans ces nouveaux écosystèmes est que tandis que les données sont hétérogénéité, il y peu ou pas de coordination entre leurs différentes unités. Ainsi, traiter l'hétérogénéité de ces données et assurer l'interopérabilité au sein de ces écosystèmes est un réel défi à relever dans le contexte actuel. Dans ce travail, notre hypothèse est que les technologies du Web Sémantique et en particulier celles des plateformes de données liées avec le standard Linked Data Platform 1.0 (LDP), peuvent être utilisées pour construire une vue homogène et un accès possible à des données auto-décrites. Ainsi, il devient moins nécessaire de contacter les fournisseurs de données pour les comprendre et les utiliser. Cependant, les outils actuels, conformes à ce standard, n'aident pas à la mise en place automatique d'écosystèmes conformes à LDP. Par conséquent, déployer de tels écosystèmes selon le standard LDP nécessite un travail important. Nous proposons ainsi une approche visant à résoudre ce problème en définissant un langage permettant de décrire l'organisations des données ainsi que leur contenus compatibles avec le standard LDP à partir de différentes sources de données et qui soient déployables sur les plateformes existantes. Nous présentons la syntaxe et la sémantique formelle de ce langage, ainsi qu'une mise en œuvre de notre approche qui se traduit part une chaine de génération et de déploiement automatique à partir d'un document de conception écrit dans ce language. Nous évaluons ce langage et notre approche au travers de plusieurs expérimentations permettant de démontrer l'automatisation de la génération de plateforme LDP à partir de sources de données existantes et hétérogènes, de la possibilité de les déployer en tenant compte de contraintes et de la possibilité de réutiliser des conceptions d'une plateforme à l'autre.