Exchanging Data amongst Semantic-Web Ontologies

On Generating Mappings and Benchmarking Data Exchange Systems

Carlos R. Rivero

University of Sevilla, Spain

Doctoral Dissertation
Supervised by Dr. David Ruiz and Dr. Rafael Corchuelo

April, 2012

First published in April 2012 by
The Distributed Group
ETSI Informática
Avda. de la Reina Mercedes s/n
Sevilla, 41012. SPAIN

http://www.tdg-seville.info
contact@tdg-seville.info

Classification (ACM 1998): D.2.12 [Interoperability]: Data mapping; H.2.5 [Heterogeneous Databases]: Data translation; H.3.4 [Systems and Software]: Performance evaluation; H.3.5 [Online Information Services]: Data sharing.

Support: Supported by the European Commission (FEDER), the Spanish and the Andalusian R&D&I programmes (grants TIN2007-64119, P07-TIC-2602, P08-TIC-4100, TIN2008-04718-E, TIN2010-21744, TIN2010-09809-E, TIN2010-10811-E, and TIN2010-09988-E).

University of Sevilla, Spain

The committee in charge of evaluating the dissertation presented by Carlos R. Rivero in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Software Engineering, hereby recommends _____ of this dissertation and awards the author the grade _____.

_____

Miguel Toro Bonilla
Catedrático de Universidad
Univ. de Sevilla

_____

Asunción Gómez Pérez
Catedrática de Universidad
Univ. Politécnica de Madrid

_____

Ernest Teniente López
Catedrático de Universidad
Univ. Politécnica de Cataluña

_____

Alberto Pan Bermúdez
Profesor Contratado Doctor
Univ. de A Coruña

_____

Carlos Pedrinaci Godoy
Research Fellow
The Open University

To put record where necessary, we sign minutes in _____, _____.

Álvaro and Cayetana exchanging Water–type Pokémon cards in their school, aged seven and three.

To my parents.

Contents

List of Figures

List of Tables

Acknowledgements

Thank you for your belief in me after so many years.
You are the reason I am here today.
A Beautiful Mind (2001), Film

A Spanish proverb says: "Gratitude is the sign of good upbringing", and, now that I am finishing my PhD student period, I cannot avoid being grateful with all the people that have supported me during these years. First, I would like to thank my professors, bosses, advisors, and, last but not least, friends Dr. Rafael Corchuelo and Dr. David Ruiz. They gave me my first opportunity to start researching in the summer of 2007 and, since the beginning, we started discussions that led to our contributions. It is obvious that it would be not possible to complete this piece of research work without your help, patience and support, thank you guys!

During these years, many colleagues have supported me with their friendship: the TDG family, the Galician guys (Alberto & Co.), and the Berliner guys (Chris & Co.). Furthermore, I am also deeply thankful to Dr. Paolo Papotti: I paid a visit to his research group in 2009 and he helped me understand the automatic generation of executable mappings in the database context, which, at the end, was the first stone of the contributions that we report in this dissertation, grazie mille Paolo!

An earlier draft of this dissertation was reviewed by the following researchers: Marcelo Arenas, Paolo Atzeni, Sören Auer, Sonia Bergamaschi, Nikos Bikakis, Angela Bonifati, Sergiu Dascalu, Dejing Dou, Jérôme Euzenat, Giorgos Flouris, Cláudio Fernando Resin Geyer, Maurizio Lenzerini, Paea LePendu, Stephen Liddle, Carlos Molina-Jiménez, Paolo Papotti, Marcelo

Soares Pimenta, Lucian Popa, Rachel Pottinger, Balder ten Cate, and Leandro Krug Wives. We are deeply indebted to them for their valuable feedback and criticisms, which helped us improve this dissertation and ongoing work.

Finally, and most important, I would like to thank my family and friends. Chiefly, I am deeply thankful to my parents since they have always helped and supported me in my pursuits; and to MJ for her infinite patience, unconditional support, and love.

Abstract

Could somebody please explain that to me like I'm a six year old?
Philadelphia (1993), Film

The goal of data exchange is to populate a target data model using data that come from one or more source data models. It is common to address data exchange building on correspondences that are transformed into executable mappings. The problem that we address in this dissertation is how to generate executable mappings in the context of semantic-web ontologies, which are the shared data models of the Semantic Web. In the literature, there are many proposals to generate executable mappings. Most of them focus on relational or nested-relational data models, which cannot be applied to our context; unfortunately, the few proposals that focus on ontologies have important drawbacks, namely: they solely work on a subset of taxonomies, they require the target data model to be prepopulated, or they interpret correspondences in isolation, not to mention the proposals that actually require the user to handcraft the executable mappings. In this dissertation, we present MostoDE, a new automated proposal to generate executable mappings in the context of semantic-web ontologies. Its salient features are that it does not have any of the previous drawbacks, it is computationally tractable, and it has been validated using a repository that has been generated using MostoBM, a benchmark that is also described in this dissertation. Our validation suggests that MostoDE is very efficient in practice and that the exchange of data it performs amongst semantic-web ontologies is appropriate.

Resumen

Por favor, ¿podría alguien explicarme esto como si tuviese seis años?
Philadelphia (1993), Película

El objetivo del intercambio de datos es cargar un modelo de datos de destino usando datos procedentes de uno o más modelos de datos fuente. Es común que el intercambio de datos sea tratado haciendo uso de correspondencias que son transformadas en mappings ejecutables. El problema tratado en esta tesis doctoral es cómo generar mappings ejecutables en el contexto de las ontologías, que son modelos de datos compartidos de la Web Semántica. En la bibliografía existen numerosas propuestas que generan mappings ejecutables. Muchas de estas propuestas se centran en los modelos de datos relacional y relacional-anidado, que no pueden ser aplicadas a nuestro contexto. Por desgracia, las pocas propuestas que se centran en ontologías tienen inconvenientes importantes, a saber: sólo trabajan con un subconjunto de las taxonomías, requieren que el modelo de datos de destino esté previamente cargado con datos, o interpretan las correspondencias de forma aislada, sin mencionar las propuestas que requieren que el usuario codifique manualmente los mappings ejecutables. En esta tesis doctoral presentamos MostoDE, una nueva propuesta automática para generar mappings ejecutables en el contexto de las ontologías de la Web Semántica. Las características más importantes de nuestra propuesta son que no posee ninguno de los inconvenientes previos, es computacionalmente tratable y ha sido validada usando un repositorio generado usando MostoBM, un benchmark que también se describe en esta tesis doctoral. Nuestra validación sugiere que MostoDE es muy eficiente en la práctica y que el intercambio de datos entre ontologías que realiza es adecuado.

Part I

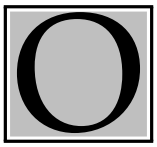
Preface

Chapter 1

Introduction

Ah, yes. The age of information (...) not, of course, that there has ever been any other kind of age. Information and knowledge: these are currencies that have never gone out of style.

Neil Gaiman, American Gods (2001)



ur goal in this dissertation is to report on our work to devise a proposal to automatically generate SPARQL executable mappings to exchange data amongst semantic-web ontologies, and a benchmark to test these executable mappings and semantic data exchange systems. The chapter is organised as follows: in Section 1.1, we first introduce the context of our research work; Section 1.2 presents the hypothesis that has motivated it and states our thesis; Section 1.4 introduces the collaborations we have conducted throughout the development of this dissertation; Section 1.3 summarises our main contributions; finally, we describe the structure of this dissertation in Section 1.5.

1.1 Research context

The goal of the Semantic Web is to endow the current Web with metadata, i.e., to evolve it into a Web of Data [84]. Currently, there is an increasing popularity of semantic-web ontologies, chiefly in the context of Linked Open Data, and they focus on a variety of domains, such as government, life sciences, geography, media, education, libraries, or scholarly publications [49]. Semantic-web ontologies build on the so-called semantic-web technologies, i.e., RDF, RDF Schema and OWL ontology languages for modelling structure and data, and the SPARQL query language to query them [7, 106, 114]. Note that, for the sake of brevity, we refer to semantic-web ontologies as ontologies in the rest of this dissertation.

Ideally, ontologies are shared data models that are developed with the consensus of one or more communities; unfortunately, reaching an agreement in a community is not a trivial task [10, 46]. Furthermore, new ontologies try to reuse existing ontologies as much as possible since it is considered a good practice; unfortunately, it is usual that existing ontologies cannot be completely reused since new parts are needed to model the data [49]. Due to these problems, there exists a variety of heterogenous ontologies to publish data on the Web, and there is a need to integrate them [49].

In the literature, there are different proposals to address this integration, such as data exchange [37], data integration [58], model matching [36], or model evolution [75]. In this dissertation, we focus on data exchange, which aims to populate a target data model using data that come from one or more source data models. Data exchange has been paid much attention in the fields of relational, nested-relational and XML data models [6, 37, 86]. Recently, with the emergence of the Semantic Web [106] and Linked Data [20], the problem is motivating many authors to work on data exchange in the context of ontologies [65, 69, 80, 87].

Without an exception, data exchange has been addressed using mappings [36, 73]. In the literature, it is common to distinguish between two kinds of mappings: correspondences and executable mappings [36, 86, 90, 95]. A correspondence is a hint that specifies which entities (or elements) in the source and target data models correspond to each other, i.e., are somewhat related [36, 90]; an executable mapping, aka operational mapping, is an executable artefact that encodes how the correspondences must be interpreted, i.e., how to perform data exchange [86, 95]. (By executable artefact we mean a SPARQL query, a Datalog rule, an XSLT script, or other means to read, transform, and output data.) Correspondences are inherently ambiguous since

there can be many different executable mappings that satisfy them, but generate different target data [4, 19, 34, 86].

Creating mappings automatically is appealing because this relieves users from the burden of writing them, checking whether they work as expected or not, making changes if necessary, and restarting this cycle [18, 83, 89, 90]. Regarding correspondences, the literature provides a number of proposals that help generate them (semi-) automatically [90], even in the context of ontologies [30, 36]. Generating executable mappings automatically in the context of relational and nested-relational data models has been studied extensively [4, 44, 86]; unfortunately, these proposals are not applicable to ontologies due to the differences amongst these data models [72, 75, 94]. This has motivated several authors to work on proposals that are specifically tailored towards ontologies, namely: Mergen and Heuser [65] presented a proposal that can deal with a subset of taxonomies only; Qin and others [87] devised a proposal that requires the target model to be pre-populated; the proposal by Mocan and Cimpian [69] interprets each correspondence in isolation, without taking their inter-relationships into account; Maedche and others [60], Parreiras and others [80], Bizer and Schultz [25], and Dou and others [34] presented a proposal that requires the user to handcraft the executable mappings.

1.2 Research rationale

In this section, we present the hypothesis that has motivated our research work in the context of generating executable mappings to exchange data amongst ontologies, and state our thesis, which we prove in the rest of the dissertation.

1.2.1 Hypothesis

Nowadays, there is an increasing interest of individuals, organisations and companies in ontologies to publish and share their data on the Web, chiefly in the context of the Linked Open Data initiative [49, 116]. According to [28], the current Web shall gradually become a Web of Data between the years 2007 and 2017, and the grand vision of the Semantic Web proposed in [106] shall be a reality in the year 2027. In the context of ontologies, information integration is one of the most important research challenges due to the variety and heterogeneity of existing ontologies [22, 25].

According to the previous argumentation, we conclude that our hypothesis is the following:

There is an increasing interest regarding ontologies and this interest is going to grow day after day. There is also a need for companies to integrate information from the Web, and the ontologies become of great importance in this context.

1.2.2 Thesis

In the context of databases, there exist a number of proposals to generate executable mappings automatically, namely: [40, 61–64, 86, 89]. They have proven to be an efficient solution to reduce integration costs when performing data exchange [44]. Furthermore, they have proven that the resulting executable mappings capture the intuition of an expert with a high precision and recall [4].

In the context of ontologies, there are also a number of proposals, namely: [25, 34, 60, 65, 69, 80, 87]. Unfortunately, they suffer from a number of drawbacks that may hinder their applicability, such as: they deal with a subset of taxonomies only, they require the target model to be pre-populated, they interpret correspondences in isolation, or they require the user to handcraft the executable mappings.

As a consequence, we conclude that our thesis is the following:

In the context of ontologies, it is possible to overcome the problems of existing proposals and develop a new one to automatically generate efficient executable mappings that capture the intuition of an expert with a high precision and recall.

1.3 Summary of contributions

To prove our thesis, we have devised MostoDE and MostoBM, which are the contributions of this dissertation, namely:

- First, MostoDE is a technique that allows to automatically generate SPARQL executable mappings to exchange data amongst ontologies, i.e., these mappings are executed by means of a query engine over the source ontology and produce data according to the target ontology. To generate them, we rely on constraints over the source and target ontologies, and correspondences between the entities of these ontologies. Note that MostoDE assumes that a set of constraints and correspondences already

exist. Our technique relies on a conceptual framework that provides a theoretically-sound foundation. They both have a number of limitations, which we think do not hinder their applicability. The limitations of our technique are that 1) it cannot deal with more than one instance of the same class or superclass (except `rdfs:Resource` and `owl:Thing`); 2) it is not able to deal with RDF collections; 3) it may generate equivalent executable mappings. The limitations of our conceptual framework are that 1) it comprises six types of constraints and four types of correspondences; 2) it is not possible to restrict more than two entities, or to establish that more than one source entity corresponds to a target entity.

Regarding this contribution, we had the following accepted publications: [41, 78, 92, 94, 95, 97]. Furthermore, we submitted an extension of MostoDE to the Knowledge and Information Systems journal.

- Second, MostoDE interprets correspondences in a way that have be to checked by domain experts and, to perform this, we rely on MostoBM, a benchmark that is able to generate real-world and synthetic data exchange problems. Furthermore, MostoBM can be used to test semantic data exchange systems, so we have devised and evaluation methodology that allows to make informed and statistically-sound decisions regarding such systems.

  Regarding this contribution, we had the following accepted publications: [32, 93, 96, 98]. Furthermore, we submitted an extension of MostoBM to the IEEE Transactions of Knowledge and Data Engineering journal.

1.4 Collaborations

Throughout the development of this dissertation, several collaborations were conducted. Not only allowed these collaborations to gather feedback regarding our research results, but they also resulted in joint publications, new research topics, and lots of interesting discussions. Below, we provide additional information about each collaboration. Further collaborations are expected in future: Dr. Paea LePendu, Stanford University (United States), Dr. Lucian Popa, IBM Research (United States), Dr. Axel Polleres, Siemens (Austria), and Dr. Sergiu Dascalu, University of Nevada (United States).

University of A Coruña (Spain): A research visit was paid to the research group of Dr. Alberto Pan, who is a professor in the Department of Information and Communication Technologies and R & D Director in Denodo Technologies, from November 24 until December 8, 2008. The goal of this visit

was to study proposals in the context of information integration using (web) databases, to present our research work, and to gather feedback.

University Roma Tre (Italy): A research visit was paid to the research group of Dr. Paolo Papotti, who is a professor in the Department of Computing and Automation, from November 1 until December 15, 2010. In this visit, we studied the proposals to automatically generate executable mappings in the context of nested-relational data models, in which Dr. Paolo Papotti is an expert with publications in major database conferences [40, 61–64, 89]. These proposals were a starting point to the research work presented in this dissertation.

Freie University Berlin (Germany): A research visit was paid to the research group of Dr. Christian Bizer, who is a professor in the Department of Information Systems and the leader of the Web-based Systems Group, from October 14 to December 5, 2011. Dr. Christian Bizer is one of the authors of the Berlin SPARQL Benchmark [24] and one of the promoters of the Linked Open Data initiative. The goal of this visit was to devise a benchmark to exchange data in the context of the Linked Open Data initiative.

The Open University (United Kingdom): Dr. Carlos Pedrinaci devised a handcrafted proposal to exchange data from OWL-S web services into MSM web services [81]. In this context, we used our automatic proposal to generate SPARQL executable mappings to perform this exchange with excellent results (see Section 10.2).

1.5 Structure of this dissertation

This dissertation is organised as follows:

Part I: Preface. It comprises this introduction and Chapter 2, in which we motivate our research work and conclude that current proposals to automatically generate executable mappings in the context of ontologies have a number of drawbacks.

Part II: Background Information. It provides information about ontologies, exchanging data, generating executable mappings, and benchmarking data exchange systems. In Chapter 3, we introduce the data and query models of ontologies. In Chapter 4, we describe the exchanging of data in current proposals. In Chapter 5, we present different proposals to

automatically generate executable mappings. In Chapter 6, we present current systems to perform data exchange and current benchmarks to test them.

Part III: Our Proposal. It reports on the core contributions we made with this dissertation. In Chapter 7, we present the conceptual framework on which our proposal is based. In Chapter 8, we present our proposal to automatically generate executable mappings in SPARQL based on constraints and correspondences. In Chapter 9, we describe our benchmark to test semantic data exchange systems. In Chapter 10, we deal with the experimental evaluation of our proposal using a repository that has been built using our benchmark.

Part IV: Final Remarks. It concludes this dissertation and highlights a few future research directions in Chapter 11.

Chapter 2

Motivation

Do... or do not. There is no try.
Star Wars: Episode V – The Empire Strikes Back (1980), Film

Although data exchange has been studied extensively, chiefly in the context of nested-relational data models, it is still necessary to solve the drawbacks of current proposals, which hinder their applicability in practice. Our goal in this chapter is to present the problems of exchanging data; to detail to which extent current proposals deal with these problems, and to motivate the need for a new proposal. The chapter is organised as follows: in Section 2.1, we introduce it; Section 2.2 presents the problems of exchanging data in detail; in Section 2.3, we conclude that none of the current proposals solve these problems at a time; Section 2.4 discusses the drawbacks of current proposals; Section 2.5 introduces our contributions and compares them with current proposals; finally, we summarise the chapter in Section 2.6.

2.1 Introduction

Nowadays, there is an increasing interest of individuals, organisations and companies in ontologies to publish, share and expose their data on the Web, chiefly in the context of (Linked) Open Data, which focuses on a variety of domains, such as government, life sciences, geography, media, education, libraries, or scholarly publications [49, 116].

Ideally, ontologies are shared data models that are developed with the consensus of one or more communities; unfortunately, reaching an agreement in a community is not a trivial task [10, 46]. Furthermore, new ontologies try to reuse existing ontologies as much as possible since it is considered a good practice; unfortunately, it is usual that existing ontologies cannot be completely reused since new parts are needed to model the data [49]. Due to these facts, there exists a variety of heterogenous ontologies to publish data on the Web, and there is a need to integrate them [10, 18, 20, 49, 74].

In the literature, there are different proposals to address the problem of integrating ontologies, and, in this dissertation, we focus on data exchange, which aims to populate a target data model using data that come from one or more source data models [65, 69, 80, 86, 87]. Unfortunately, these proposals have a number of drawbacks that hinder their applicability in practice. Consequently, it is still necessary to research on exchanging data in the context of ontologies, which is our purpose in this dissertation.

2.2 Problems

Exchanging data is not a trivial task and, if it is not performed appropriately, it may increase integration costs. In this section, we present the problems that must be coped with by proposals to perform data exchange, with an emphasis on the ontology context. These problems are the following:

(P1) To handcraft executable mappings: Data exchange is usually a complex task that requires a strong knowledge of the domain of the data exchange problem, so domain experts are required to perform it. Executable mappings, which consist of rules or queries that can be executed over a reasoner or a query engine to perform data exchange, help reduce integration costs. To handcraft executable mappings implies that domain experts have to code them, which is usually not trivial. Furthermore, it is not surprising that existing data models may undergo modifications in response to requests for change, therefore, when handcrafting executable

mappings, software engineers have to adapt and maintain them according to these modifications. As a conclusion, the automatic generation of executable mappings is appealing to reduce integration costs.

(P2) To rely on many target data: The goal of the data exchange is to populate a target data model with data that come from source data models. The target data model is not usually expected to be pre-populated. Therefore, due to the nature of the problem of exchanging data, relying on many target data may increase integration costs, since, if the target data model cannot be pre-populated with actual data, software engineers must provide representative-enough synthetic data. As a conclusion, it is not appealing to rely on many target data when performing data exchange.

(P3) To not benchmark the exchanged data: Without an exception, systems that are suitable to perform data exchange have uneven performance. Therefore, it is appealing to assess and rank them from an empirical point of view. As a conclusion, it is necessary to devise benchmarks and statistical evaluation methodologies to make informed and statistically-sound decisions about such systems.

(P4) To not check the interpretation of correspondences: Data exchange proposals construct target data that may not be as expected by domain experts. This is crucial in the context of executable mappings, which encode how correspondences between source and target data models must be interpreted. Correspondences are inherently ambiguous since there can be many different executable mappings that satisfy them, but generate different target data. As a conclusion, it is necessary to check if the interpretation of correspondences that a proposal assumes agrees with the interpretation of domain experts.

(P5) To interpret correspondences in isolation: A correspondence is a hint that specifies which entities in the source and target data models correspond to each other, i.e., are somewhat related. Since they are hints, data exchange proposals that rely on correspondences must not interpret them in isolation, i.e., without taking their inter-relationships into account. For instance, an instance of property $sch{:}name$, which denotes the name of a person, cannot be exchanged in isolation, but in the context of an instance of class $sch{:}Person$, which denotes a person. As a conclusion, executable mappings must not interpret correspondences in isolation but in the context of other inter-related correspondences.

(P6) To encode non semantic-web executable mappings: In the context of nested-relational data models, executable mappings are encoded using

XQuery or XSLT, which build on the structure of the XML documents on which they are executed. However, to exchange data amongst ontologies, these executable mappings must be encoded in a language that is independent from the structure of the documents used to represent an ontology, since the same ontology may be serialised to multiple languages, e.g., XML, N3, or Turtle. As a conclusion, in the context of ontologies, it is mandatory to use executable mappings encoded using appropriate query or rule languages, such as SPARQL, or SWRL, since they are specifically tailored towards ontologies.

(P7) To not deal with arbitrary graphs: An ontology is a shared data model that represents an arbitrary graph in which there may be zero, one or more paths connecting two arbitrary entities, there may not exist a unique root entity, and there may be cycles. As a conclusion, it is a must to deal with arbitrary graphs to exchange data amongst ontologies.

(P8) To not deal with arbitrary taxonomies: Ontologies allow to define arbitrary taxonomies amongst their entities, i.e., it is possible to define that a class is a subclass of another class, or that a property is a subproperty of another property. Furthermore, these taxonomies may be completely arbitrary in the sense that all relations are allowed, such as an undefined number of specialisation levels, a class (property) may have more than one superclass (superproperty), or a class (property) is subclass (subproperty) of itself. As a conclusion, it is necessary to deal with arbitrary taxonomies when performing data exchange in the context of ontologies.

Problems P1, P2, P3, P4, and P5 deal with common problems of exchanging data amongst data models of any type. Problems P6, P7, and P8 focus on performing data exchange amongst ontologies.

2.3 Analysis of current solutions

Fagin and others [37] and Arenas and Libkin [9] presented the theoretical foundations for performing data exchange using executable mappings in the context of relational and XML data models. It is important to notice that relational data models are a special case of nested-relational data models [86] and that a nested-relational data model is defined by means of a tree that comprises a number of nodes, which may be nested and have a number of attributes, and it is also possible to specify referential constraints that relate these attributes.

In the context of nested-relational data models, data exchange has been studied extensively [3, 40, 44, 63, 86, 89]. Popa and others [86] devised the

state-of-the-art proposal regarding data exchange in nested-relational data models. Although it seems efficient and effective enough to be used in practical applications, it cannot be applied in the context of ontologies. The main reason is that it builds on computing primary paths from the root node of a data model to every attribute in this model, and then applying a variation of the well-known Chase algorithm that takes referential constraints into account [33]. In general, an ontology is not a tree, but a graph in which there is not a root node, which prevents us from computing primary paths, and it can contain cycles, which prevents us from using the Chase algorithm. Note that there are more differences between nested-relational data models and ontologies [72, 75, 94], such as an instance in a nested-relational data model has a unique type that corresponds to an existing node, contrarily, an ontology instance may have multiple types of several existing classes, which need not be related by specialisation; in addition, in nested-relational data models, queries to exchange data are encoded using XQuery or XSLT, contrarily, in an ontology, these queries must be encoded in a language that is independent from the structure used to represent an ontology.

These differences amongst nested-relational data models and ontologies have motivated several authors to work on proposals that are specifically tailored towards ontologies. The initial work on data exchange in this context put the emphasis on defining the problem and the strategies to address it, which mainly consisted of devising handcrafted correspondences and performing data exchange using ad-hoc proposals [60, 76]. Later proposals performed data exchange using reasoners instead of ad-hoc proposals [34, 69, 104], or using SPARQL query engines [80, 85, 94].

The majority of current ontology proposals to exchange data in the literature focus on the automatic generation of correspondences and executable mappings that interpret them; there are a few proposals, however, that require the user to provide executable mappings using ad-hoc languages. Correspondences can be handcrafted using visual tools [2, 89, 91], or discovered automatically using model matching proposals [30, 36, 90]. This dissertation focuses on the automatic generation of executable mappings to perform data exchange, finding them is orthogonal to the problem of generating executable mappings, which is the reason why we do not discuss them further in this section. The most closely related proposals in the field of ontologies were presented by Mergen and Heuser [65], Qin and others [87], Mocan and Cimpian [69], Maedche and others [60], Parreiras and others [80], Bizer and Schultz [25], and Dou and others [34]. Table 2.1 summarises how these proposals deal with the problems of performing data exchange (see Section 2.2).

Mergen and Heuser [65] devised an automated proposal that works with

Proposals	P1	P2	P3	P4	P5	P6	P7	P8
Bizer and Schultz [25]	×	✓	×	×	✓	✓	✓	✓
Dou and others [34]	×	✓	×	×	✓	✓	✓	✓
Maedche and others [60]	×	✓	×	×	✓	✓	✓	✓
Mergen and Heuser [65]	✓	✓	×	×	✓	×	×	×
Mocan and Cimpian [69]	✓	✓	×	×	×	✓	✓	✓
Parreiras and others [80]	×	✓	×	×	×	✓	✓	✓
Popa and others [86]	✓	✓	×	×	✓	×	×	×
Qin and others [87]	✓	×	×	×	✓	✓	✓	✓

P1 = To handcraft executable mappings; P2 = To rely on many target data; P3 = To not benchmark the exchanged data; P4 = To not check the interpretation of correspondences; P5 = To interpret correspondences in isolation; P6 = To encode non semantic-web executable mappings; P7 = To not deal with arbitrary graphs; P8 = To not deal with arbitrary taxonomies.

Table 2.1: *Comparison of current proposals to exchange data.*

a subset of taxonomies in which there are only classes, data properties, and single specialisations amongst classes. Their algorithm analyses every class correspondence independently, and tries to find the set of correspondences that are involved in its properties, super-classes, and super-properties; this helps identify data that must be exchanged together and the many possible exchanges that can be performed. These subsets of correspondences are then translated into an ad-hoc script language that was devised by the authors.

Qin and others [87] devised a semi-automatic proposal that relies on data-mining. They first require the user to select a subset of source and target data for each data property; these are used to feed a mining algorithm that attempts to discover a set of queries that can exchange these data; these queries are then sorted according to an ad-hoc metric, and the top ones are selected and transformed into Web-PDDL, Datalog, or SWRL rules. The most important problem with this proposal is that it requires the target data model to be pre-populated so that the data-mining algorithm can work, which is not the case in common data exchange problems [4, 18, 37, 86]; if the target cannot be pre-populated with actual data, then the user must provide representative-enough synthetic data. Furthermore, it requires the user to select appropriate subsets of data from both the source and target data models, i.e., subsets of data that capture variability well enough; this is not a trivial task since the data have

to be selected very carefully to avoid over-fitting, i.e., generating executable mappings that can deal with only the selected training data.

Mocan and Cimpian [69] studied the problem of data exchange in the context of semantic-web services. They presented a formal framework to describe correspondences in terms of first-order logic formulae that can be mapped onto WSML rules very easily. Their proposal is similar in spirit to the one by Maedche and others [60], whose focus was on modelling correspondences in a general-purpose setting. The main difference with the previous proposals is that Mocan and Cimpian went a step beyond formalising correspondences and devised a mediator that executes them using a WSML reasoner. Note that no attempt is made to identify groups of correspondences that must be taken into account together, which may easily lead to incoherent target data, i.e., data that does not satisfy the constraints in the target ontology [4, 19, 86].

Parreiras and others [80] presented a proposal within the framework of Model-Driven Engineering. They extended the ATL metamodel to support OWL 1 ontologies, which allows to express constraints on them using the OCL language. They devised a mapping language called MBOTL by means of which users can express executable mappings that are later transformed into SPARQL and Java by means of a library of ATL transformations. Their proposal does not build on correspondences, but requires users to handcraft and maintain their executable mappings using MBOLT. This is similar in spirit to the proposals by Bizer and Schultz [25] and Dou and others [34]; the difference is the language used to represent the executable mappings.

Regarding benchmarks, Alexe and others [4] devised a benchmark that is used to test proposals to automatically generate executable mappings to perform data exchange in the context of nested-relational data models. It provides eleven data exchange patterns that occur frequently in the information integration context. To the best of our knowledge, this is the unique benchmark in the literature specifically tailored towards data exchange so it allows to construct data exchange scenarios, each of which comprises a source and a target data model and a set of queries to perform data exchange. As a conclusion, it can be used to construct repositories to check if the interpretation of correspondences is according to domain experts. Unfortunately, this benchmark is not suitable in the context of ontologies due to the differences with respect to nested-relational data models (see above).

There are a number of benchmarks to test systems that implement semantic-web technologies. Bizer and Schultz [24] and Schmidt and others [103] focused on comparing the performance of SPARQL query engines; Guo

and others [42] focused on comparing the performance of RDF stores, reasoners, and query engines; Wu and others [117] focused on the performance of an implementation of a reasoner in Oracle.

In addition, there are a number of proposals that deal with patterns in the ontology evolution context, which may be used to test the interpretation of correspondences. Stojanovic and others [109] identified sixteen atomic changes an ontology may undergo, and some combinations of these changes are very frequent in practice, so this motivated the authors to devise a catalogue of twelve common composite changes. Noy and Klein [75] extended the catalogue of atomic changes to twenty two, and they classified these changes according to whether data are lost after changes are applied or not.

2.4 Discussion

The previous proposals have problems that hinder their applicability in practice: Popa and others [86] focused on nested-relational data models (P6, P7, and P8), Mergen and Heuser [65] dealt with only a subset of taxonomies (P7 and P8), Qin and others [87] required the target model to be pre-populated (P2), Mocan and Cimpian [69] interpreted correspondences in isolation (P5), Maedche and others [60], Parreiras and others [80], Bizer and Schultz [25], and Dou and others [34] relied on handcrafted executable mappings (P1). Furthermore, none of these proposals use a repository to test whether or not they interpret correspondences as expected by domain experts (P4).

Regarding benchmarks, none of the existing benchmarks focus on data exchange problems (P3), i.e., they do not provide source and target ontologies and mechanisms to exchange data [24, 42, 103, 117]. In addition, they are domain-specific, which entails that they provide ontologies with a fixed structure in a particular domain, so they only allow to tune the construction of synthetic data but not their structure. Last, they focus on SELECT queries instead of the CONSTRUCT queries that are required to exchange data.

Regarding patterns, current proposals focus on the ontology evolution context [75, 109]. Unfortunately, the specification of the evolution of an ontology does not take how data are exchanged into account (P3). Our catalogue of synthetic patterns summarises common changes we have found in real-world information integration problems: not only specify they how the structure of the source ontology evolves, but also how source data must be exchanged by means of SPARQL queries of the CONSTRUCT type. Our synthetic patterns are instantiated by tuning a number of structure parameters that allow to construct synthetic scenarios. These scenarios range from simple atomic changes to complex composite changes.

2.5 Our proposal

To solve the previous drawbacks, we present a proposal called MostoDE to automatically generate SPARQL executable mappings in the context of ontologies, which are executed by means of a query engine to exchange data (see Chapters 7 and 8). Contrarily to the previous proposals, ours is able to compute the executable mappings automatically, which saves the user from the burden of designing them (P1). Our proposal is based on constraints of the source and target ontologies, and correspondences amongst them (P2).

We have devised MostoBM, a benchmark to test semantic data exchange systems, and an evaluation methodology that allows to make informed and statistically-sound decisions regarding such systems (P3) (see Chapters 9 and 10). The interpretation of correspondences that our proposal assumes has been tested by means of a publicly-available repository that comprises four real-world data exchange scenarios and $3,780$ synthetic data exchange scenarios (P4) (see Chapter 10). To construct these synthetic data exchange scenarios, we have used MostoBM that provides a tool to construct synthetic data exchange scenarios by means of a number of parameters to scale them.

Furthermore, our proposal identifies groups of correspondences that must be analysed together, which impedes the resulting executable mappings from producing incoherent target data (P5). It generates executable mappings that are represented in standard SPARQL, which allows to perform data exchange using a commodity SPARQL engine (P6). Our proposal can deal with quite a complete subset of ontologies that adhere to the OWL 2 Lite profile, which allows for classes, data properties, object properties, and multiple specialisations between classes and properties (P7) (see Appendix A). Last, our proposal does not require the target data model to be pre-populated, and it does not require the user to provide any additional data to compute the resulting executable mappings (P8).

2.6 Summary

In this chapter, we have motivated the reason for this piece of research work. We have analysed the problems of exchanging data and current proposals in the literature to exchange data, and we have concluded that none of these proposals solves these problems at a time. This motivates the necessity of our contribution and that it advances the state of the art a step forward.

Part II

Background Information

Chapter 3

Semantic-web Ontologies

You do not want to argue semantics with a PhD candidate.
The Rundown (2003), Film

The Semantic Web aims to evolve the current Web into a Web of Data that revolves around ontologies. In this chapter, we introduce the data models and query languages to define and query ontologies, the current state of the Web of Data, and the principles of the Linked Data initiative. The chapter is organised as follows: in Section 3.1, we introduce it; Sections 3.2 and 3.3 present the data model and the query model, respectively; in Section 3.4, we present the current state of the Web of Data; Section 3.5 describes the principles of the Linked Data initiative; finally, we summarise the chapter in Section 3.6.

3.1 Introduction

The vision of the Semantic Web is an evolution of the old Web, which mainly comprises a variety of documents devised and consumed by humans, to a new Web that should be automatically manipulated by machines [16]. To achieve this, the goal of the Semantic Web is to endow the current Web with metadata, i.e., to evolve it into a Web of Data [84, 106]. Ontologies are the data models of this Web of Data; they build on the semantic-web technologies, i.e., RDF, RDF Schema and OWL ontology languages for modelling structure and data, and the SPARQL query language to query them [7, 114].

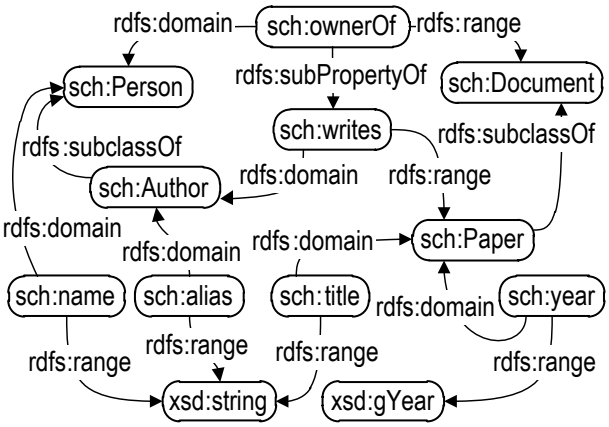
Currently, there is an increasing popularity of the Web of Data, chiefly in the context of (Linked) Open Data in the following domains: government, life sciences, geography, media, education, libraries, or scholarly publications [49]. Linked Data is a successful initiative of the Web of Data that consists of a number of principles to publish, connect, and query data in the Web that rely on semantic-web technologies [20, 22, 49].

The Web of Data can be used as it was a large database, i.e., it allows to answer structured queries from users [84]. In this context, the most important research challenge is to cope with scalability problems of processing amounts of data at Web scale [21]. Furthermore, the Web of Data is been constructed on demand, as an ongoing effort that involves different parties [10].

Some authors have also proposed that the Web of Data should follow a pay-as-you-go approach, i.e., it does not require full integration to provide useful services [39, 100]. In this context, the effort of integrating different sources is not made in one single initial step, but it is a continuous effort that involves different parties [49, 100]. Usually, when a pay-as-you-go system is deployed for the first time, it can process only simple queries, but, as it evolves, it is expected to be able to process more complex queries.

3.2 The data model

In this section, we report on the OWL 2 Lite profile ontology language, and how it is represented by means of the RDF language. Herein after, wherever we mention OWL, we are just implicitly referring to this version unless otherwise stated. The OWL ontology language allows to model both the structure and the data of an ontology. Regarding modelling structure, an OWL ontology comprises classes, data properties, and object properties, each of which is identified by a URI. Figures 3.1(a) and 3.1(b) show the structure of the same



(a) *Structure (Graph).*

sch:Person
sch:name <xsd:string>
sch:ownerOf <sch:Document>

sch:Document

sch:Author [sch:Person]
sch:alias <xsd:string>
sch:writes [sch:ownerOf] <sch:Paper>

sch:Paper [sch:Document]
sch:year <xsd:gYear>

(b) *Structure (tree).*

sch:CRRivero [sch:Author]
sch:name **"Carlos Rivero"**
sch:alias **"Carlos R. Rivero"**
sch:alias **"Carlos R. Osuna"**
sch:writes [**sch:CIKMPaper**]
sch:writes [**sch:ERPaper**]

sch:CIKMPaper [sch:Paper]
sch:year **"2011"**

sch:ERPaper [sch:Paper]

(c) *Data.*

```
(sch:Person,     rdf:type,        owl:Class)
(sch:Author,     rdf:type,        owl:Class)
(sch:Author,     rdfs:subClassOf, sch:Person)
(sch:CIKMPaper,  rdf:type,        sch:Document)
(sch:CIKMPaper,  rdf:type,        sch:Paper)
(sch:CIKMPaper,  sch:year,        "2011"^^xsd:gYear)
```

(d) *RDF.*

Figure 3.1: *Example of the structure and data of an ontology.*

sample ontology using graph-based and tree-based notations, respectively. In this dissertation we use both notations but, for the sake of simplicity, we use the tree-based notation in this section since it is easier to read than the graph-based notation.

In Figure 3.1(b), sch:Author is a class that models an author, and we denote it as a circle. In this example, we use namespace sch: as a prefix. A class can be specialised into other classes, e.g., sch:Author is a subclass of sch:Person, and we denote it as sch:Author [sch:Person]. An example of a data property is sch:name, which models the name of a person, and we denote it as a square. Data properties have a set of classes as domain and a basic XML data type as range, e.g., the domain of sch:name is {sch:Person}, and we denote it by nesting sch:name into sch:Person. The range of sch:name is xsd:string, and we denote it as sch:name<xsd:string>. An example of an object property is sch:writes, which models "an author writes a paper", and we denote it as a pentagon. Object properties have a set of classes as domain and range, e.g., the domain of sch:writes is {sch:Author} and the range is {sch:Paper}. Data and object properties can be also specialised into other properties, e.g., sch:writes is subproperty of sch:ownerOf, and we denote it as sch:writes [sch:ownerOf].

Beyond subclass, domain, range and subproperty constructs, the OWL Lite profile ontology language provides other constructs to represent other constraints, e.g., owl:sameAs, which deals with relating two instances that model the same real-world object; owl:minCardinality, which restricts the minimal number of property instances that an ontology should contain; or owl:versionInfo, which is devised to express meta-information.

Regarding data modelling, a class instance is identified by its own URI, and it may have a number of types (see Figure 3.1(c)). We denote a class instance as a diamond, e.g., sch:CRRivero is a class instance of type sch:Author. sch:CRRivero is also, implicitly, an instance of sch:Person, since sch:Author is a subclass of sch:Person; reasoners are used to make this knowledge explicit [112]. In addition, it is possible to have class instances of types that are not related by specialisation, e.g., assume that sch:CRRivero is the URL of a web page that provides the biography of a person, therefore, sch:CRRivero might have both types sch:Person and sch:Document. A data property instance relates a class instance with a literal, and we denote it as a square, e.g., sch:name relates sch:CRRivero with "Carlos Rivero". An object property instance relates two class instances, e.g., sch:writes relates sch:CRRivero and sch:CIKMPaper, which represents that an author has written a particular paper. By default, data and object property instances have a minimal

cardinality of zero, e.g., there is no data property instance of sch:year relating sch:ERPaper and the actual year of the paper; however, there is a data property instance of sch:year relating sch:CIKMPaper and "2011". Furthermore, by default, data and object property instances may be multiple, e.g., sch:CRRivero is related to "Carlos R. Rivero" and "Carlos R. Osuna" by data property sch:alias.

RDF, which is based on triples, is used to store both the structure and data of an OWL ontology. A triple comprises three elements: the first one is called subject, the second one is called predicate, and the third one is called object. In Figure 3.1(d), we present some examples of RDF triples.

3.3 The query model

In this section, we report on the SPARQL 1.1 query language. Herein after, wherever we mention SPARQL, we are just implicitly referring to this version unless otherwise stated. SPARQL queries are used to retrieve and/or construct triples. They are based on triple patterns that are similar to triples, but they allow to specify variables in the subject, predicate, and/or object, which are prefixed with a '?'. A set of triple patterns match RDF data by unifying the variables of the triple patterns with these data.

SPARQL provides four types of queries, namely: SELECT, ASK, DESCRIBE, and CONSTRUCT. SELECT and ASK queries can only retrieve data. A SELECT query comprises a unique WHERE clause, which specifies a set of triple patterns, and a set of variables to be returned, and it retrieves a plain table of values. An ASK query comprises a single WHERE clause and it checks whether the specified triple patterns have an instantiation in the RDF data.

DESCRIBE and CONSTRUCT are able to retrieve and construct RDF data. A DESCRIBE query comprises a unique WHERE clause and a set of variables to be returned. This structure is similar to SELECT queries but, instead of returning a plain table of values, DESCRIBE queries retrieve an RDF graph, so the client needs not know the structure of the retrieved RDF graph.

A CONSTRUCT query comprises two parts: the WHERE and CONSTRUCT clauses. The triple patterns of the WHERE clause are used to retrieve data from an RDF store, and the triple patterns of the CONSTRUCT clause are responsible for constructing RDF data. CONSTRUCT queries return a unique RDF graph, which is formed by taking each query solution, replacing the variables of the WHERE clause, and combining the resulting triples by set union.

```
Q1: CONSTRUCT {
  ?p rdf:type   sch:Person .
} WHERE {
  ?p rdf:type   sch:Author . }
```

```
Q2: CONSTRUCT {
  ?p rdf:type   sch:Person .
  ?p sch:name  ?n .
} WHERE {
  ?p rdf:type   sch:Author .
  ?p sch:alias   ?a .
  BIND(toName(?a) AS ?n) }
```

Figure 3.2: *Example of queries.*

In this dissertation, we focus on CONSTRUCT queries, since they are suitable to perform data exchange amongst ontologies. Figure 3.2 shows two examples of SPARQL queries, in which we use a notation to give names to queries (Q_1 and Q_2) since they help us refer to them. In this example, Q_1 retrieves instances of sch:Author (the WHERE clause) and reclassifies those instances as sch:Person (the CONSTRUCT clause); Q_2 reclassifies instances of sch:Author as sch:Person, and transforms the value of sch:alias into sch:name by means of a toName function using a BIND clause, which assigns the final value to variable ?n.

3.4 Current state of the Web of Data

The Web is being transformed from a Web of documents into a global data space called the Web of Data, which consists of a huge graph that comprises billions of units of data [22, 84]. There exists a variety of ontologies that focus on a variety of domains, such as government, life sciences, geography, media, education, libraries, or scholarly publications [49].

Ontologies are growing steadily, chiefly in the context of Linked Open Data: in 2007, there were roughly 12 such ontologies and, as of the time of writing this dissertation, there exist 326 ontologies in this context, according to [59]. These ontologies are represented in the Linked Open Data cloud (see Figure 3.3), and some illustrative examples of them are the following:

- Gene Ontology [13]: it is a daily updated ontology that represents gene and gene product attributes, which comprises roughly 32,000 classes with a dozen specialisation levels.
- DBpedia [23]: it is a community effort to annotate and make the data stored at Wikipedia accessible by means of an ontology. DBpedia 3.7

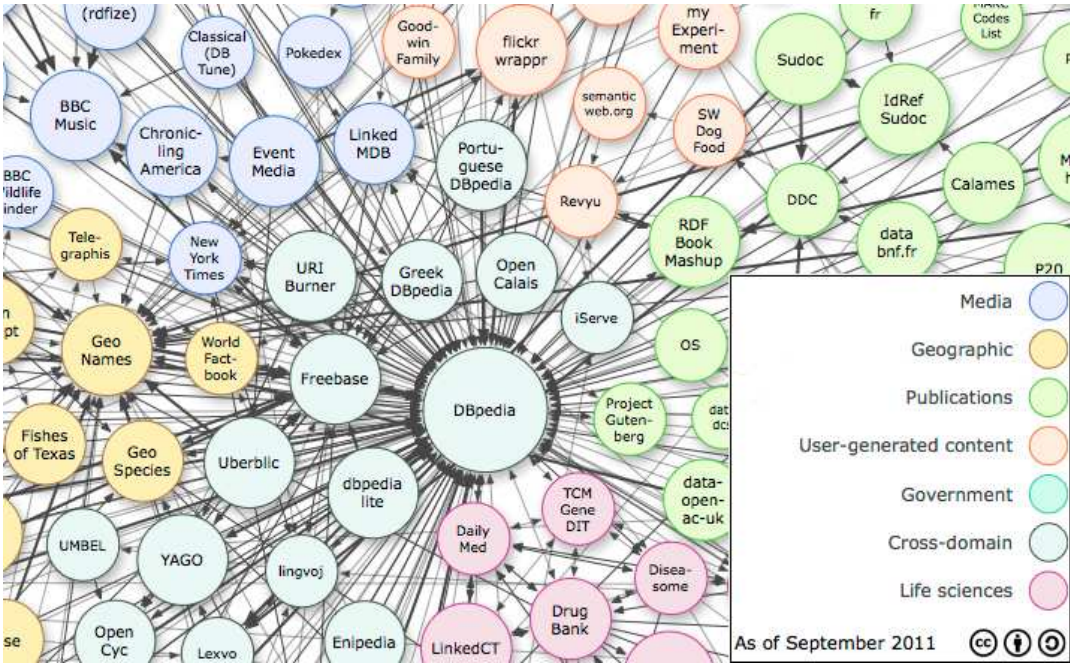


Figure 3.3: *A part of the Linked Open Data cloud [59].*

comprises 3.64 million individuals, i.e., instances of owl:Thing, using 97 different natural languages.

- BBC Programmes [56]: the BBC provides this ontology to publicise the programmes they broadcast in both radio and television. It uses the Programmes Ontology that describes programmes including brands, series, and episodes.

- LinkedGeoData [12]: it is an effort to add a spatial dimension to the Web of Data that uses the data of the OpenStreetMap project, whose goal is to construct an editable map of the world. It comprises data about 350 million places and 30 million ways.

3.5 Principles of Linked Data

Linked Data is a successful initiative of the Semantic Web that is based on four principles to publish, connect, and query data in the Web that rely on semantic-web technologies [49]. The intuitive idea behind these principles

is to apply the general architecture of the Web to share structured data on a global scale. All principles refer to URIs as a mechanism to identify things uniquely, i.e., resources on the Web that include real-world objects, such as people, authors, books, or geographic places, or abstract concepts, such as relationships that state that an author has written a book, or that an author is a person.

The first principle states that things should be identified using URIs. The second principle promotes using HTTP URIs to allow users to look things up. The third principle establishes that we should use W3C specifications (RDF, RDF Schema, OWL, and SPARQL) to represent and query things. Finally, the fourth principle states that related things should be explicitly linked; this is accomplished by using the following OWL constructs: owl:sameAs, owl:equivalentClass, and owl:equivalentProperty.

3.6 Summary

In this chapter, we have presented an overview of the Web of Data. We have described the languages on which the Web of Data builds, i.e., the data model (RDF and OWL) and the query model (SPARQL). Furthermore, we have summarised current state of ontologies by focusing on four examples. Last, we have described the principles of the Linked Data initiative.

Chapter 4

Exchanging Data

Data! data! data! –he cried impatiently–
I can't make bricks without clay.

Sir Arthur Conan Doyle, The Adventure of the Copper Beeches (1892)

Data exchange aims to populate a target data model using data that come from one or more source data models. In this chapter, we present the applications of exchanging data and how it is performed in current proposals in the literature. The chapter is organised as follows: in Section 4.1, we introduce it; Section 4.2 describes several applications of exchanging data; Sections 4.3 and 4.4 deal with the whole process of performing data exchange amongst ontologies based on reasoners and query engines, respectively; in Section 4.5, we present mappings, which are the cornerstone components to perform data exchange; finally, we summarise the chapter in Section 4.6.

4.1 Introduction

Integration is a common term by means of which researchers refer to a variety of problems, including: data integration, which deals with answering queries posed over a target data model using a number of source data models only [45, 58]; model matching, aka model mapping, whose goal is to unveil correspondences amongst the elements of two data models [36, 90]; model merging, which consists of automatically generating an integrated model by means of multiple and possibly overlapping input models [48, 74]; model evolution, which focuses on the modifications of an existing data model in response to a request for change [38, 75]; or data exchange, which aims to populate a target data model using data that come from one or more source data models [9, 34, 37, 65, 69, 80, 86, 87].

This dissertation focuses on data exchange, which has been paid much attention in the fields of relational and nested-relational data models [9, 37, 86]. In the ontology context, there exists a variety of heterogenous ontologies to publish data on the Web, and it is necessary to exchange data amongst them, which is motivating many authors to work on this topic [34, 65, 69, 80, 87].

In addition to ad-hoc proposals [76], there are several automated proposals to perform data exchange that rely on reasoners [104] and query engines [86]. When using ad-hoc proposals, data exchange is based on handcrafted pieces of software that transform source data into target data. When using reasoners, data exchange consists of reclassifying source instances into target instances by means of rules. Finally, when using queries, data exchange is performed by executing a number of queries that extract data from the source data models, transform them, and load the results into the target data model.

4.2 Applications

Data exchange is a mainstream integration proposal by itself [9, 37, 86, 94]. However, it is used in other contexts, namely:

- Data integration: a query is posed over a target data model and has to be answered using source data models only. The target query is reformulated into a single query over source data models, which is divided into individual queries that retrieve data from each source data model. Furthermore, an execution plan is generated that specifies how these source queries must be executed. Finally, the source data retrieved must be combined into data that conform to the target data model, which is performed by means of data exchange.

- Model evolution: a data model may undergo changes due to a variety of reasons. After the data model has changed, data have to be updated to adapt them according to the new data model. Data exchange can be used to update these data. In this case, we assume that the source data model is the data model before changes are applied, and the target data model is the data model after changes are applied.

- Model merging: two or more source data models are combined to create a new target data model, which includes information about all source data models. In this context, source data must be transformed into target data according to the new target model. Data exchange can be used to perform this transformation.

- Vocabulary adaptation: usually, two ontologies offer the same data but structured according to different vocabularies. Therefore, we wish to adapt the vocabulary of the source ontology to the vocabulary of the target ontology. Data exchange can be used to perform this adaptation.

- Publication of Linked Data: before the success of the Linked Open Data initiative, there were a variety of ontologies that publish their data without taking Linked Data principles into account. Usually, there is a need for these ontologies to publish their data using these principles. Data exchange can be used to perform this publication.

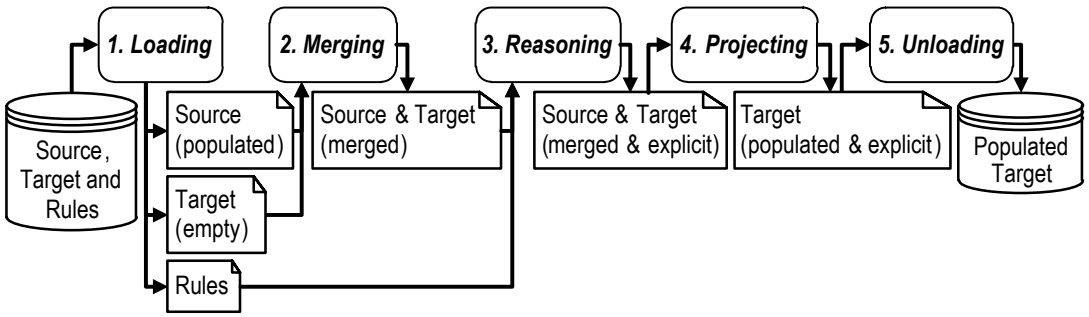
The previous applications provide an overall idea of how important data exchange is becoming. They also make it clear to foresee that this research field shall remain active in the years to come.

4.3 The reasoner-based process

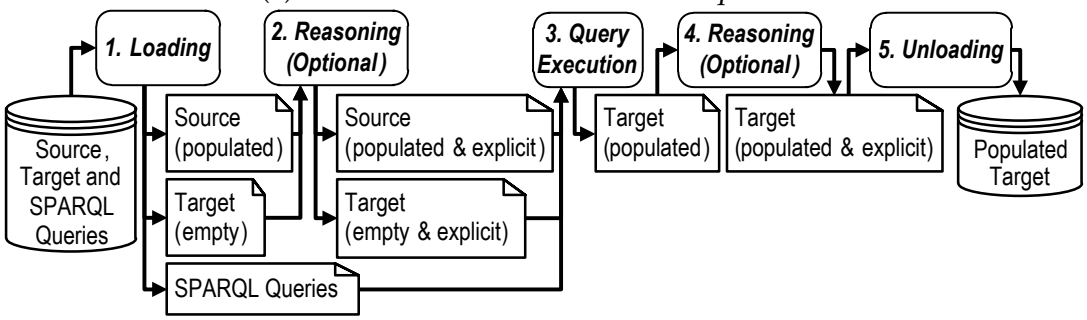
In the context of ontologies, exchanging data by means of reasoners consists of reclassifying source instances into target instances by means of rules. This process comprises five steps (see Figure 4.1(a)), namely:

i. Loading: This step consists of loading the source and target ontologies and the set of rules from a persistent storage into the appropriate internal data structures.

ii. Merging: In this step, source and target ontologies are merged into a single ontology, since the vast majority of reasoners take only single ontology as input.



(a) *Workflow of the reasoner-based process.*



(b) *Workflow of the query-based process.*

Figure 4.1: *Workflows of data exchange.*

iii. Reasoning: In this step, the reasoner is responsible for applying rules to reclassify source instances into target instances. Furthermore, built-in rules are applied to make it explicit the knowledge in the source and target ontologies.

iv. Projecting: This step deals with the splitting of the merged ontology by projecting the structure and data of the target ontology.

v. Unloading: This step deals with saving the target ontology from the internal data structures to a persistent storage.

4.4 The query-based process

In the context of ontologies, performing data exchange using a query engine consists of executing a number of queries that extract data from the source

data models, transform them, and load the results into the target data model. This process comprises five steps (see Figure 4.1(b)), namely:

i. Loading: This step consists of loading the source and target ontologies and the set of SPARQL queries from a persistent storage into the appropriate internal data structures.

ii. Reasoning over source: This step is optional and deals with making it explicit the knowledge in the source ontology. This step is not required if the knowledge is already explicit in the source ontology, but it is a must in many cases since SPARQL does not deal with RDF Schema or OWL entailments.

iii. Query execution: This step consists of executing a set of SPARQL queries over the source ontology to produce instances of the target ontology. The result of this step must be the same regardless of the order in which queries are executed.

iv. Reasoning over target: This step is also optional and it deals with making it explicit the knowledge in the target ontology.

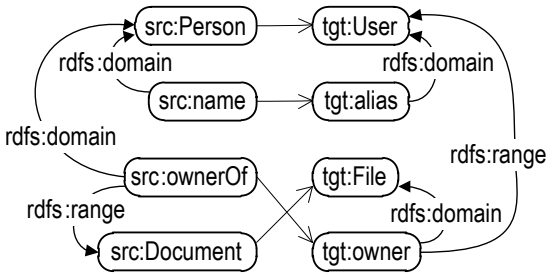
v. Unloading: This step deals with saving the target ontology from the internal data structures to a persistent storage.

4.5 Mappings

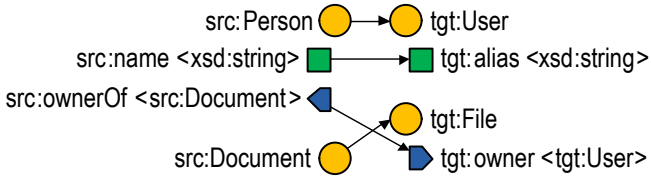
Without an exception, the cornerstone components of proposals that exchange data are the so-called mappings, which describe the relationships amongst source and target data models in different ways [17]. Mappings provide the "semantic glue" that is needed to perform data exchange amongst data models. In the literature, it is common to distinguish between two kinds of mappings: correspondences and executable mappings [30, 36, 86, 90, 94].

We use a sample data exchange problem to illustrate correspondences and executable mappings (see Figure 4.2). In this problem, we assume that we want to exchange the source ontology, which models people and documents, into the target ontology, which models users and files. Note that, for the sake of simplicity, we omit the ranges of data properties when using the graph-based notation.

On the one hand, a correspondence is a hint that specifies which entities (or elements) in the source and target data models correspond to each other,



(a) *Graph-based notation.*



(b) *Tree-based notation.*

Figure 4.2: *An example of a data exchange problem.*

```
Q1: CONSTRUCT {                 | Q2: CONSTRUCT {
  ?d rdf:type   tgt:File .      |   ?p rdf:type    tgt:User .
} WHERE {                       |   ?p tgt:alias   ?n .
  ?d rdf:type   src:Document . } | } WHERE {
                                |   ?p rdf:type    src:Person .
                                |   ?p src:name   ?n . }
```

Figure 4.3: *Example of executable mappings in SPARQL.*

i.e., are somewhat related [30, 36, 90]. We denote a correspondence as an arrow between a source and a target entity (see Figures 4.2(a) and 4.2(b)). Sample correspondences in our examples are the following: between classes src:Person and tgt:User, src:Document and tgt:File, between data properties src:name and tgt:alias, and between object properties src:ownerOf and tgt:owner.

On the other hand, an executable mapping, aka operational mapping, is an executable artefact that encodes how the correspondences must be interpreted, i.e., how to perform data exchange [86, 94]. Note that, by executable artefact, we mean a SPARQL query, a Datalog rule, an XSLT script, or other

means to read, transform, and output data.

Figure 4.3 shows two sample executable mappings in SPARQL that encodes an interpretation of the previous correspondences. Query Q_1 encodes the interpretation of the correspondence between classes src:Document and tgt:File, which entails that every instance of class src:Document is copied as class tgt:File. Furthermore, Q_2 encodes the interpretation of the correspondence between src:Person and tgt:User, which entails that every instance of class src:Person is copied as class tgt:User, and the correspondence between src:name and tgt:alias, which entails that every object of each instance of property src:name is copied as the object of an instance of property tgt:alias, and the subject of the latter instance has tgt:User type.

4.6 Summary

In this chapter, we have presented an overview of data exchange, which aims to populate a target data model using data that come from one or more source data models. We have summarised a number of applications of data exchange, and we have described the process of exchanging data amongst ontologies using reasoners and query engines. Finally, we have introduced mappings, which are the cornerstone components of data exchange proposals.

Chapter 5

Generating Mappings

I don't need a map! I have the GPS.
Never need a map again, thank you.
Cars (2006), Film

To reduce integration costs, it is desirable to automatically generate mappings to perform data exchange. In this chapter, we present current proposals in the literature to automatically generate them. The chapter is organised as follows: in Section 5.1, we introduce it; Sections 5.2 and 5.3 introduce several forms in the literature to represent and generate correspondences, respectively; furthermore, Sections 5.4 and 5.5 describe several forms to represent and generate executable mappings, respectively; finally, we summarise the chapter in Section 5.6.

5.1 Introduction

It is well-known that generating and maintaining handcrafted mappings increases integration costs [18, 83, 89]. With the goal of reducing them, there are a number of proposals in the literature to automatically generate mappings, i.e., correspondences and executable mappings. This generation may be fully-automated, in which case the user has no intervention, or may be semi-automated, in which case a visual tool helps the user generate them.

Since we focus on the automatic generation of mappings, mechanisms to represent them are needed to make them machine-processable. On the one hand, there are several proposals to represent correspondences and, in general, almost every proposal uses its own representation format. On the other hand, executable mappings are represented in several rule languages in the context of reasoner-based proposals. However, in query-based proposals, they are usually represented by means of SPARQL, since it is the unique language recommended by the W3C for querying ontologies.

5.2 Representing correspondences

Regarding nested-relational data models, Popa and others [86] used the most simple form of correspondences: a logic equality between a source entity and a target entity. These correspondences are represented in a internal model that is similar to nested-relational data models. Mecca and others [63] used a more general form of correspondences: they relate a number of source entities with a target entity via a transformation function, and they use the same internal model as Popa and others [86].

Regarding ontologies, Bouquet and others [26] presented the C-OWL ontology, which allows to contextualise ontologies, i.e., express constraints that are not present in ontologies but are needed for certain tasks. The C-OWL ontology represents oriented correspondences from source entities to target entities. Five types of relation predicates are allowed, namely: more general, more specific, equivalent, disjoint and overlap. C-OWL allows to express correspondences amongst classes, properties, and class instances.

Euzenat [35] proposed the alignment format to represent correspondences. This format is divided into three levels, each of which characterises the type of correspondences. The first level relates a pair of entities in the modelling language by using a relation predicate, which can be equivalence, subsumption, incompatibility or even fuzzy relations. The second level is a generalisation of

the first level allowing sets or lists of entities. Finally, the third level considers expressions in a given language such as first-order logic, SWRL or SPARQL.

Maedche and others [60] devised the Semantic Bridge Ontology, which allows to relate classes, properties, and class instances by means of semantic bridges. They identify five dimensions (aspects) for correspondences, namely: entity, cardinality, constraint, transformation and structural. The entity dimension establishes which entities are related. The cardinality dimension covers how many entities are related. The constraint dimension are conditions that must hold to when integrating the source and target data models. The transformation dimension specifies how the data have to be exchanged. Finally, the structural dimension covers how the semantic bridges are combined. These combinations include specialisation, abstraction, composition and alternative.

Miles and Bechhofer [66] developed SKOS that stands for Simple Knowledge Organisation System, and it was developed for sharing and linking the knowledge of organisation systems in the Web. SKOS provide the class skos:Concept, which represents a concept in the domain of interest. These concepts can be aggregated by means of concept schemata. Furthermore, SKOS provides semantic relations and mapping properties, which allow to express correspondences amongst two or more concepts by means of constructs such as narrower, broader or related.

Mocan and Cimpian [69] proposed a formal framework to describe correspondences in terms of first-order logic formulae, each of which relates a source and a target entity. Furthermore, Serafini and Tamilin [104] used the OWL 1 ontology language to represent correspondences, each of which relates two classes (more general or more specific) or two class instances.

Scharffe and Fensel [101], Scharffe and others [102] proposed a number of patterns to represent correspondences inspired in the design patterns in the software engineer context. The authors have devised a pattern library, in which each pattern comprises a name, the problem that this patterns aims to solve, the solution, and consequences, i.e., known uses and other related patterns. They classify their patterns into generic patterns, such as subsumption, conditional, or transformation, and application-specific patterns, in which they focus in correspondences amongst relational databases and ontologies.

5.3 Generating correspondences

The literature provides several proposals that help generate correspondences (semi-) automatically by means of matchers. This research topic has

been extensively studied during the last fifteen years in both contexts: nested-relational data models and ontologies. In this dissertation, we focus on two well-known surveys in both contexts [36, 90]. Recent surveys on this topic were provided by Bellahsene and others [15] and Shvaiko and Euzenat [107].

Rahm and Bernstein [90] focused on nested-relational data models and they classified matchers into two groups: individual and combined matchers. The former generate correspondences based on a single criterion, and the latter generate correspondences by combining multiple matching criteria. Regarding individual matchers, they classified them according to the following features:

- Instance vs. data model: matchers may take instances, data models or both into account.
- Entity vs. structure: matchers may use entities in isolation or combinations of them.
- Linguistic vs. constraints: matchers may be based on a linguistic proposal, such as names and textual descriptions of entities, or based on constraints of the data models.

Regarding combined matchers, on the one hand, hybrid matchers apply different matching criteria in the same algorithm; on the other hand, composite matchers combine the results of different (individual or hybrid) matchers.

Euzenat and Shvaiko [36] focused on ontologies and they extended the classification of Rahm and Bernstein [90] by introducing features such as the following:

- Syntactic vs. external vs. semantic: syntactic matchers are based on the source and target entities, external matchers use auxiliary resources, e.g., human input or a thesaurus of relationships, and semantic matchers use some formal semantics.
- Refined structure: structure matchers can be graph-based, taxonomy-based, based on repositories of data models, model-based, or based on data analysis and statistics.
- Kind of input: matchers can be terminological, structural, semantics, or extensional if the input is formed by strings, entities, data models (entities and constraints), or instances, respectively.

Finally, there exist some visual tools in the literature that help generate handcrafted correspondences. In the context of nested-relational data models, there are several tools that allows to handcraft these correspondences using visual languages [44, 62, 64, 89]. These tools also incorporate matching services to help designers generate correspondences for large data models. Therefore, the process of generating handcrafted correspondences becomes a best-effort process: matchers are used to generate simple correspondences amongst the data models and, then, the user is responsible for refining simple correspondences and building more complex correspondences with the help of these visual tools.

5.4 Representing executable mappings

In the literature, there are several languages to define executable mappings to perform data exchange. In the context of relational data models, SQL or Datalog with extensions have been used to represent these mappings [8, 37, 111]. In the context of nested-relational data models, XSLT or XQuery have been used to represent them [3, 40, 63, 86, 89].

In the context of ontologies, Mergen and Heuser [65] relied on ad-hoc scripts to represent executable mappings, which are formed by groups of correspondences with optional conditions of when to apply them. Furthermore, there exist a number of proposals that use different rule languages to represent these mappings, such as Datalog, Web-PPDL, SWRL or WSML [69, 87]. These rules have to be executed by means of a reasoner to perform data exchange.

There are other proposals that use SPARQL to represent executable mappings. Bizer and Schultz [25] devised a mapping language called R2R based on a subset of SPARQL 1.0 that provides three data properties, namely: r2r:prefixDefinitions, r2r:sourcePattern and r2r:targetPattern. The former property allows to define the prefixes used in a particular executable mapping as a single string. The goal of the other properties is to define the triple patterns of the WHERE and CONSTRUCT clauses, respectively, each one as a single string. In addition to these properties, R2R offers other properties to define transformation functions, to reduce the scope of the executable mappings to particular ontologies, or to specify provenance information.

Parreiras and others [80] used an extension of the OCL language to represent executable mappings, and these intermediate mappings are automatically transformed into executable mappings represented in SPARQL 1.0. Polleres and others [85] presented preliminary ideas on the use of SPARQL executable

mappings to perform data exchange. They focused on the limitations of the SPARQL 1.0 query language to work as a language to describe executable mappings, and they proposed a number of extensions, such as regular expressions to describe paths, external functions, or aggregations. Some of these extensions have been incorporated to the SPARQL 1.1 working draft. Furthermore, Ressler and others [91] devised a visual language to represent executable mappings that are automatically transformed into SPARQL 1.0.

5.5 Generating executable mappings

Generating executable mappings automatically in the context of relational and nested-relational data models has been studied extensively [40, 44, 63, 86, 89]. Unfortunately, the proposals in this context are not applicable to ontologies due to the differences with respect to nested-relational data models [72, 75, 94]. Some examples of these differences are the following: a nested-relational data model represents a tree, whereas an ontology represents an arbitrary graph; an instance in a nested-relational data model has a unique type whereas an instance may be of multiple types in an ontology; or the query language to encode executable mappings in the context of nested-relational data models cannot be applied to ontologies. This has motivated several authors to work on proposals that are specifically tailored towards ontologies [25, 34, 65, 69, 80, 87, 91].

There exists a few proposals that require the user to handcraft executable mappings [25, 34, 80, 91]. Amongst these proposals, we focus on [80, 91] since they provide mechanisms to automatically generate executable mappings, but they do not build on correspondences.

Parreiras and others [80] presented a proposal within the framework of Model-Driven Engineering. They extended the ATL metamodel to support OWL 1 ontologies, which allows to express constraints on them using the OCL language. They devised a mapping language called MBOTL by means of which users can express executable mappings that are later transformed into SPARQL and Java by means of a library of ATL transformations.

Ressler and others [91] devised a visual tool to define executable mappings that are automatically translated into SWRL rules. It allows to define correspondences between entities or individuals of the source and target ontologies. Furthermore, it provides mechanisms to express logical, string manipulation, and mathematical functions. To test their tool, the authors used a case study in the maritime domain, which is included in the geospatial domain.

Regarding automatic proposals, Mergen and Heuser [65] devised a proposal to automatically generate executable mappings that works with a subset of taxonomies in which there are classes, data properties, and single specialisations amongst classes. Their algorithm analyses every class correspondence independently, and tries to find the set of correspondences that are involved in its properties, superclasses, and superproperties; this helps identify data that must be exchanged together and the many possible exchanges that can be performed. These subsets of correspondences are then translated into an ad-hoc script language that was devised by the authors.

Mocan and Cimpian [69] studied the problem of data exchange in the context of semantic-web services. They presented a formal framework to describe correspondences in terms of first-order logic formulae that can be mapped onto WSML rules very easily. Their proposal is similar in spirit to the one by Maedche and others [60], whose focus was on modelling correspondences in a general-purpose setting. The main difference with the previous proposals is that Mocan and Cimpian [69] went a step beyond formalising correspondences, which are transformed into WSML rules, and devised a mediator that executes these rules using a WSML reasoner.

Qin and others [87] devised a semi-automatic proposal that relies on data-mining. They first require the user to select a subset of source and target data for each data property; these are used to feed a mining algorithm that attempts to discover a set of queries that can exchange these data; these queries are then sorted according to an ad-hoc metric, and the top ones are selected and transformed into Web-PDDL, Datalog, or SWRL rules. This proposal uses a module to determine whether two instances in different ontologies represent the same real-world entity.

5.6 Summary

In this chapter, we have presented several languages to represent correspondences, such as the Semantic Bridge Ontology, C-OWL or SKOS, and a classification of proposals to automatically generate them. Furthermore, we have described several languages to represent executable mappings, such as XQuery, SWRL or SPARQL, and several proposals in the literature to handcraft or automatically generate them.

Chapter 6

Benchmarking Data Exchange Systems

I have no data yet. It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts.

Sir Arthur Conan Doyle, A Scandal in Bohemia (1891)

Current systems that implement semantic-web technologies have uneven performance when performing data exchange. In this chapter, we introduce the benchmarks and patterns in the literature to test the performance of data exchange systems. The chapter is organised as follows: in Section 6.1, we introduce it; Section 6.2 introduces current systems that implement semantic-web technologies; in Section 6.3, we present current benchmarks in the literature to test the performance of data exchange systems; Section 6.4 describes patterns in the literature that can be used to test data exchange systems; finally, we summarise the chapter in Section 6.5.

6.1 Introduction

Currently, there are a large number of systems that implement semantic-web technologies and are, thus, suitable to perform data exchange amongst semantic-web ontologies. Without an exception, these systems have uneven performance [24, 42, 103, 117], which makes assessing and ranking them from an empirical point of view appealing, since this helps make informed decisions about which the best system for a particular integration problem is.

In the literature, there exists a number of benchmarks to assess and rank the performance of such systems. These benchmarks incorporate catalogues of common patterns that usually occur in information integration and/or evolution contexts. In this chapter, we present these benchmarks and patterns, and current systems that implement semantic-web technologies that are suitable to perform data exchange.

In the context of executable mappings, correspondences are inherently ambiguous since there can be many different executable mappings that satisfy them, but generate different target data [4, 19, 34, 86]. Since executable mappings encode a particular interpretation of correspondences, it is a must to check whether or not this interpretation agrees with the interpretation of domain experts [4, 19, 34, 86]. Data exchange benchmarks can be used to check this interpretation since they incorporate information on how data is exchanged, so the final target data using a benchmark and a set of executable mappings should be the same.

6.2 Current systems

Currently, there exists a variety of systems that implement semantic-web technologies. These systems are suitable to exchange data amongst ontologies, e.g., TDB, Pellet, ARQ, Jena, Oracle, OWLIM, Sesame, or Virtuoso to mention a few; as of the time of writing this dissertation, the W3C lists a total of 272 systems [115]. A semantic data exchange system comprises an RDF store, a reasoner, and a query engine, which is only needed when data exchange is implemented using query engines. The systems that implement semantic-web technologies provide different services, e.g., TDB is an RDF store, Pellet is a reasoner, ARQ is a query engine, Jena provides an RDF store and a reasoner, and Oracle or OWLIM provide an RDF store, a reasoner and a query engine.

In the rest of this section, we describe the systems that we use in this dissertation. TDB 0.8.7 [110] is a high performance, transactional RDF store based on

files. TDB runs on both 32-bit and 64-bit Java Virtual Machines, but it is specifically tailored towards 64-bit systems since they can provide more memory for file caching. Pellet 2.2.2 [82] is a reasoner for Java that supports every OWL profile and implements a number of optimisations to improve its performance, such as incremental reasoning, reasoning over XML Schema datatypes, or axiom pinpointing, which provides a justification when any arbitrary rule is applied to an ontology [108].

ARQ 2.8.7 [11] is a SPARQL query engine that natively supports the SPARQL 1.0 specification. Furthermore, it provides a number of extensions to support the SPARQL 1.1 working draft, such as the following:

- Free text search: ARQ is combined with Apache Lucene to perform free text searches over RDF data.
- Property paths: this extension allows to add paths over the graphs defined by RDF data, which are specified by means of regular expressions.
- Assignments: this extension allows to explicitly define a value for a particular variable of a triple pattern. This value may be any expression, including constants, other variables, or functions applied to constants or variables.
- Custom functions: ARQ provides a function library that includes string manipulation, math utilities, or manipulation of collections. Furthermore, it allows to extend this library by means of user-defined functions that are implemented in Java.

Jena 2.6.4 [53] offers an in-memory RDF store that provides an API to load and unload RDF files in several formats, such as RDF/XML, N3, or Turtle. Furthermore, it offers an in-memory reasoner that allows four types of reasoning, namely:

- Transitive: this type of reasoning only implements the transitive and reflexive features of `rdfs:subPropertyOf` and `rdfs:subClassOf`.
- RDF Schema: this type implements a configurable subset of the RDF Schema constructs.
- OWL: this type implements a configurable subset of the OWL 1 Lite and OWL 1 Full profiles.
- Generic: this type of reasoning allows to incorporate user-defined rules.

owl:allValuesFrom	owl:inverseOf
owl:differentFrom	owl:onProperty
owl:disjointWith	owl:Restriction
owl:equivalentClass	owl:sameAs
owl:equivalentProperty	owl:someValuesFrom
owl:FunctionalProperty	owl:SymmetricProperty
owl:hasValue	owl:TransitiveProperty
owl:InverseFunctionalProperty	

Table 6.1: *OWL constructs of the OWLSIF set of rules.*

Oracle 11.2.0.1.0 [77] offers an RDF store, a reasoner, and a query engine as part of Oracle Spatial 11g, which is an installation option for Oracle Database 11g Enterprise Edition. The RDF store is based on a normalised, compressed and partitioned storage architecture that is able to manage the repetition of URIs and literals across triples in real-world ontologies. The reasoner provides three different set of rules to make it explicit the knowledge, namely:

- RDFS++: this set of rules comprises all constructs of RDF Schema, i.e., rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain, and rdfs:range. It also incorporates two constructs of OWL, namely: owl:sameAs, and owl:InverseFunctionalProperty.
- OWLSIF: this set of rules was proposed by ter Horst [112], and it extends RDF Schema constructs with the OWL constructs shown in Table 6.1.
- OWL RL: this set of rules corresponds to the OWL RL profile, which is "aimed at applications that require scalable reasoning without sacrificing too much expressive power" [71].

The query engine natively supports the use of SQL to query ontologies. Furthermore, it natively supports SPARQL queries of the SELECT type that are automatically translated into SQL queries. For the rest of SPARQL queries, this query engine relies on ARQ to support them (see below).

OWLIM 4.2 [79] offers an RDF store, a reasoner, and a query engine. It is divided into three versions: 1) Lite, which deals with tens of million triples; 2)

SE, which is able to handle tens of billion triples; and 3) Enterprise, which provides a replication cluster based on OWLIM SE. In this dissertation, we focus on OWLIM 4.2 Lite, which is completely implemented in Java. Regarding the reasoner, it allows to use the set of rules of RDF Schema, the ones proposed by ter Horst [112], and the rules of OWL RL profile. In addition, this reasoner supports user-defined rules. The query engine of OWLIM 4.2 supports the SPARQL 1.1 query language.

6.3 Current benchmarks

There exists a number of benchmarks to help software engineers make informed decisions about different systems. Guo and others [42] presented LUBM, a benchmark to compare systems that support semantic-web technologies, which is based on the context of universities. This benchmark provides a single ontology, a data constructor algorithm that allows to create scalable synthetic data, and fourteen SPARQL queries of the SELECT type.

Wu and others [117] presented the conclusions regarding an implementation of an inference engine for Oracle. This engine implements the RDF Schema and OWL entailments, i.e., it performs reasoning over OWL and RDF Schema ontologies. Furthermore, they present a performance study based on two examples: LUBM, which comprises data that ranges from 6.7 to 133.7 million triples, and UniProt, which comprises five million triples roughly.

Bizer and Schultz [24] presented BSBM, a benchmark to compare the performance of SPARQL queries using native RDF stores and SPARQL-to-SQL query rewriters. Their benchmark focuses on an e-commerce case study, and they provide a data constructor and a test driver: the former allows to create large ontologies and offers RDF and relational outputs to compare the proposals; the latter emulates a realistic workload by simulating multiple clients that concurrently execute queries. The benchmark consists of twelve SPARQL queries that are divided into ten SELECT queries, one DESCRIBE query, and one CONSTRUCT query.

Schmidt and others [103] presented SP^{2}Bench, a benchmark to test SPARQL query management systems, which is based on DBLP and comprises both a data constructor and a set of benchmark queries in SPARQL. They study the DBLP dataset to construct a realistic set of synthetic data by measuring probability distributions for certain attributes, e.g., authors or cites in articles or papers. The benchmark queries comprise seventeen queries that are divided into fourteen SELECT queries and three ASK queries.

Furthermore, there are other benchmarks that are specifically tailored towards data exchange problems. Alexe and others [4] devised a benchmark that is used to test data exchange systems in the context of nested-relational models. This benchmarks allows to scale the structure of source and target data models, and the data of the source data model when performing data exchange using a set of parameters.

6.4 Data exchange patterns

There are a number of proposals that identify common patterns that usually occur in integration contexts. On the one hand, there are proposals based on nested-relational models. On the other hand, there are other proposals that focus on the ontology evolution context. These patterns can be used to test the interpretation of a set of correspondences by executable mappings.

Regarding nested-relational proposals, Alexe and others [4] devised a benchmark that comprises eleven data exchange patterns that occur frequently in the information integration context. Their benchmark focuses on the evaluation of mapping systems for nested-relational models, which consist of visual programming systems to assist a designer in the generation of mappings between two data models. Examples of the scenarios of this benchmark are the Object Fusion scenario, which deals with the merging of a number of unrelated source data models into a unique target data model, or the Manipulation of Atomic Values scenario, which consists of transforming one source element into various target elements and vice versa.

Regarding ontology evolution proposals, ontology evolution can be seen as a data exchange problem in which the source is the original ontology, and the target is an evolution of the source ontology. Stojanovic and others [109] identified sixteen atomic changes an ontology may undergo, e.g., adding or removing classes, subclasses, properties, subproperties, property domains or property ranges. These changes can be seen as the simplest operations building on which the evolution of an ontology may be specified. Due to the fact that some combinations of changes are very frequent in practice, this motivated the authors to devise a catalogue of twelve common composite changes that are expressed at a higher level, such as merging classes, extracting subclasses, moving properties, or copying classes.

Noy and Klein [75] studied the differences between database and ontology evolution. Based on these differences, they extended the catalogue of atomic changes to twenty two, accounting for, e.g., the reclassification of an

instance as a class or viceversa, the declaration of two classes to be disjoint, moving properties to a subclass or superclass, or moving classes amongst a subclass taxonomy. Furthermore, they classified these changes according to whether data are lost after changes are applied or not: in nine out of twenty two changes data are not lost, in nine out of twenty two changes data may be lost, and in the rest of the changes data are not lost if they are transformed according to the new ontology.

6.5 Summary

In this chapter, we have presented an overview on the benchmarking of data exchange systems. We have presented current systems that implement semantic-web technologies. Then, we have analysed current benchmarks in the literature to test systems that implement semantic-web technologies. Finally, we have analysed a number of patterns that usually occur in integration contexts, which may be used to test the interpretation of a set of correspondences by executable mappings.

Part III

Our Proposal

Chapter 7

Conceptual Framework

Are you kidding? I model everything (...).
The modeling is like pushing and pulling clay.
William Gibson, Spook Country (2007)

To describe our proposal, we rely on a conceptual framework that models ontologies, entities, triples, or executable mappings to mention a few. Our description relies on a small subset of the Z standard. The chapter is organised as follows: in Section 7.1, we introduce it; Section 7.2 presents the foundations of this framework; Sections 7.3, 7.4, 7.5, 7.6 describe how we model triples, ontologies, executable mappings, and data exchange problems, respectively; Sections 7.7 and 7.8 deal with the satisfaction of constraints and correspondences, respectively; Section 7.9 highlights a number of limitations of this framework; finally, we summarise the chapter in Section 7.10.

7.1 Introduction

Our proposal relies on a conceptual framework. In this chapter, we define its foundations, i.e., entities (classes, data properties, and object properties), URIs, literals, and blank nodes. We also define triples, each of which is a three-element tuple that comprises a subject, a predicate, and an object. Special types of triples are constraints and correspondences: the former have a constraint construct as its predicate, whereas the latter have a correspondence construct as predicate.

Furthermore, we define ontologies, which comprises a number of entities and a number of constraints over these entities. Triple patterns generalise the concept of triples by allowing variables in the subject, the predicate, and/or the object. Executable mappings comprise two sets of triples patterns, i.e., the set of triple patterns in the CONSTRUCT clause, and the set of triple patterns in the SELECT clause, respectively. A data exchange problem comprises two ontologies (the source and the target), and a set of correspondences between the entities of these ontologies. Last, we define when a constraint and a correspondence are satisfied.

In our formulation, we use a minimal subset of the Z mathematical language standard (ISO/IEC 13568:2002) [52]. This helps us make sure that our specifications are correct using the appropriate tools. This standard has been used, e.g., to describe the W3C WSDL specification [29].

7.2 Foundations

Entities lay at the heart of every ontology, and they are denoted by means of URIs. Entities can be classified into classes and properties; the latter can be further classified into data properties and object properties. We denote the sets of classes, data properties, and object properties as follows:

$$[Class, DataProperty, ObjectProperty]$$

and define the set of entities and properties as follows:

$$Entity == Class \cup Property$$
$$Property == DataProperty \cup ObjectProperty$$

Ontologies also build on a set of URIs, which are used to identified resources on the Web, literals, which denote values of simple data types, and blank nodes, which denote anonymous data. We denote these sets as follows:

$[URI, Literal, BlankNode]$

Note that $Entity$ is a subset of URI, and URI, $Literal$ and $BlankNode$ are pairwise disjoint sets [55]:

$$\begin{array}{l} Entity \subseteq URI \\ (URI \cap Literal = \emptyset) \land (URI \cap BlankNode = \emptyset) \land \\ \quad (Literal \cap BlankNode = \emptyset) \end{array}$$

7.3 Triples

Triples are three-tuples in which the first element is called subject, the second predicate, and the third object. They help describe both the structure and data of ontologies. We define the set of all triples as follows:

$$\begin{aligned} Triple &== Subject \times Predicate \times Object \\ Subject &== URI \cup Literal \cup BlankNode \\ Predicate &== Property \cup BuiltInConstruct \\ Object &== URI \cup Literal \cup BlankNode \end{aligned}$$

Note that there is a contradiction between the RDF and the SPARQL specifications. On the one hand, the RDF specification does not allow to use literals in the subject of a triple [55]. On the other hand, the SPARQL specification actually allows to use literals in the subject of a triple [47], therefore, we are able to build triples in which the subject is a literal using SPARQL. Since our research focuses on SPARQL queries, we have decided to include literals in the subject of triples.

Note, too, that a predicate can be a property or a built-in construct. By built-in construct, we mean a URI that denotes one of the predefined RDF, RDF Schema, and Mosto constructs with which we deal. (In the sequel we use prefix mosto: to refer to the constructs that we have defined in our proposal.) We define these sets as follows:

$$\begin{aligned} BuiltInConstruct &== \{\texttt{rdf:type}\} \cup \\ &\quad\ ConstraintConstruct \cup \\ &\quad\ CorrespondenceConstruct \\ ConstraintConstruct &== \{\texttt{rdfs:subClassOf}, \\ &\quad\ \texttt{rdfs:subPropertyOf}, \\ &\quad\ \texttt{rdfs:domain}, \\ &\quad\ \texttt{rdfs:range}, \end{aligned}$$

$$\begin{aligned} & \quad \text{mosto:strongDomain}, \\ & \quad \text{mosto:strongRange}\} \\ \text{CorrespondenceConstruct} == & \{\text{mosto:classToClass}, \\ & \quad \text{mosto:dataToData}, \\ & \quad \text{mosto:objectToObject}, \\ & \quad \text{mosto:dataToClass}\} \end{aligned}$$

ConstraintConstruct does not actually include every possible construct in the OWL Lite profile, but the minimal subset of constraints with which we deal. Section 7.7 provides additional details regarding the semantics of this subset of constraints, and Appendix A provides additional details on how to translate other constructs into this minimal subset.

Set CorrespondenceConstruct includes the correspondence constructs with which we can deal. They allow to establish correspondences from classes to classes, data properties to data properties, object properties to object properties, and data properties to classes. Section 7.8 provides additional details regarding the semantics of these constructs.

For the sake of convenience, we also define the following subsets of triples:

$$\begin{aligned} \text{Constraint} &== \text{Subject} \times \text{ConstraintConstruct} \times \text{Object} \\ \text{Correspondence} &== \text{Subject} \times \text{CorrespondenceConstruct} \times \text{Object} \end{aligned}$$

7.4 Ontologies

An ontology is a representation of a data model, which comprises a description of the structure of the data, and the data themselves. For the purpose of this dissertation, we just need the description of the structure. An ontology can thus be defined by means of a two-tuple, which consists of a set of entities and a set of constraints. We then define the set of all ontologies as follows, where $\mathbb{P}$ denotes powerset:

$$\begin{aligned} \text{Ontology} == \{ & E : \mathbb{P}\,\text{Entity}; C : \mathbb{P}\,\text{Constraint} \mid \\ & \forall s : \text{Subject}; p : \text{ConstraintConstruct}; o : \text{Object} \bullet \\ & (s, p, o) \in C \Rightarrow \{s, o\} \subseteq E\} \end{aligned}$$

For the sake of convenience, we define the following projection functions:

$$\begin{aligned} & \text{entities} : \text{Ontology} \rightarrow \mathbb{P}\,\text{Entity} \\ & \text{constraints} : \text{Ontology} \rightarrow \mathbb{P}\,\text{Constraint} \\ \hline & \forall E : \mathbb{P}\,\text{Entity}; C : \mathbb{P}\,\text{Constraint}; o : \text{Ontology} \mid o = (E, C) \bullet \\ & \quad \text{entities}(o) = E \land \\ & \quad \text{constraints}(o) = C \end{aligned}$$

7.5 Executable mappings

In our proposal, executable mappings are SPARQL conjunctive queries of the CONSTRUCT type. Such queries consist of a CONSTRUCT clause that specifies which triples have to be constructed in the target ontology, and a WHERE clause that specifies which triples should be retrieved from the source ontology. These clauses can then be represented as sets of triple patterns that are implicitly connected by means of a logical and. A triple pattern generalises the concept of triple by allowing the subject, the predicate, and/or the object to be variables. We denote the set of all variables as follows:

$$[Variable]$$

and define the set of all triple patterns as follows:

$$\begin{aligned} TriplePattern &== Subject^? \times Predicate^? \times Object^? \\ Subject^? &== Subject \cup Variable \\ Predicate^? &== Predicate \cup Variable \\ Object^? &== Object \cup Variable \end{aligned}$$

An executable mapping can thus be represented as a two-tuple in which the first component corresponds to the set of triple patterns in the CONSTRUCT clause, and the second component corresponds to the set of triple patterns in the WHERE clause. We then define the set of all executable mappings as follows:

$$ExecutableMapping == \mathbb{P}\, TriplePattern \times \mathbb{P}\, TriplePattern$$

For the sake of convenience, we define an instance of a class as a triple pattern of the form: $(s, \text{rdf:type}, c)$, in which $s \in Subject^?; c \in Class$; and an instance of a property as a triple pattern of the form: (s, p, o), in which $s \in Subject^?$, $p \in Property$, and $o \in Object^?$. Furthermore, we define the following projection functions:

$$\begin{aligned} &subject : TriplePattern \rightarrow Subject^? \\ &predicate : TriplePattern \rightarrow Predicate^? \\ &object : TriplePattern \rightarrow Object^? \\ \hline &\forall s : Subject^?; p : Predicate^?; o : Object^?; t : TriplePattern \mid t = (s, p, o) \bullet \\ &\quad subject(t) = s \land \\ &\quad predicate(t) = p \land \\ &\quad object(t) = o \end{aligned}$$

Note that $Triple \subseteq TriplePattern$, which implies that instances may refer to both triples and triple patterns, and that the previous projection functions can be applied to both triples and triple patterns.

7.6 Data exchange problems

A data exchange problem consists of a source ontology, a target ontology, and a set of correspondences between some of their entities. We define the set of all data exchange problems as follows:

$$\begin{aligned} DataExchangeProblem == \{o_1, o_2 : Ontology; V : \mathbb{P}\, Correspondence \mid \\ \forall v : Correspondence \bullet v \in V \Rightarrow \\ subject(v) \in entities(o_1) \land \\ object(v) \in entities(o_2)\} \end{aligned}$$

For the sake of convenience, we define the following projection functions:

$$\begin{aligned} &source : DataExchangeProblem \to Ontology \\ &target : DataExchangeProblem \to Ontology \\ &correspondences : DataExchangeProblem \to \mathbb{P}\, Correspondence \\ \hline &\forall o_1, o_2 : Ontology; V : \mathbb{P}\, Correspondence; d : DataExchangeProblem \mid \\ &\quad d = (o_1, o_2, V) \bullet \\ &\qquad source(d) = o_1 \land \\ &\qquad target(d) = o_2 \land \\ &\qquad correspondences(d) = V \end{aligned}$$

7.7 Satisfaction of constraints

Given an executable mapping, it is important to know whether it satisfies a given constraint or not. This prevents us from retrieving data that does not satisfy the constraints in the source or the target of a data exchange problem. In the following subsections, we describe how to infer whether a set of triple patterns in the CONSTRUCT or the WHERE clause of an executable mapping satisfies a given constraint or not. We use symbol $\models$ to denote satisfaction, T to denote a set of triple patterns, c, c_1, and c_2 to denote classes, and p, p_1, and p_2 to denote properties.

7.7.1 Subclassification constraints

A set of triple patterns satisfies a subclass or a subproperty constraint if, for every instance of a given class or property, we may find another instance of the corresponding subclass or subproperty, respectively. The following inference rules formalise this idea:

$$[\text{SC}]\ \frac{\forall s : \text{Subject}^? \bullet (s, \text{rdf:type}, c_1) \in T \Rightarrow (s, \text{rdf:type}, c_2) \in T}{T \models (c_1, \text{rdfs:subClassOf}, c_2)}$$

$$[\text{SP}]\ \frac{\forall s : \text{Subject}^?; o : \text{Object}^? \bullet (s, p_1, o) \in T \Rightarrow (s, p_2, o) \in T}{T \models (p_1, \text{rdfs:subPropertyOf}, p_2)}$$

7.7.2 Domain and range constraints

A set of triple patterns satisfies a domain or a range constraint regarding a property if, for every property instance that states that a subject is related to an object, there are additional class instances that state that the type of the subject or the object are the appropriate classes, respectively. The following inference rules formalise this idea:

$$[\text{D}]\ \frac{\forall s : \text{Subject}^?; o : \text{Object}^? \bullet (s, p, o) \in T \Rightarrow (s, \text{rdf:type}, c) \in T}{T \models (p, \text{rdfs:domain}, c)}$$

$$[\text{R}]\ \frac{\forall s : \text{Subject}^?; o : \text{Object}^? \bullet (s, p, o) \in T \Rightarrow (o, \text{rdf:type}, c) \in T}{T \models (p, \text{rdfs:range}, c)}$$

7.7.3 Strong domain and strong range constraints

We deal with two additional constraints to which we refer to as strong domain (mosto:strongDomain) and strong range (mosto:strongRange). Intuitively, if a class is the strong domain of a property, that means that this property has cardinality one regarding that class; similarly, if a class is the strong range of a property, that means that for every instance of that class, there must be a subject that is related to that instance by means of the property. These semantics are expressed in OWL as a combination of several triples. For the sake of simplicity, we introduced mosto:strongDomain and mosto:strongRange as shorthands (see Appendix A for further details). The following inference rules formalise the satisfaction of these constraints:

$$[\text{SD}]\ \frac{\forall s : \text{Subject}^? \bullet (s, \text{rdf:type}, c) \in T \Rightarrow \exists o : \text{Object}^? \bullet (s, p, o) \in T}{T \models (c, \text{mosto:strongDomain}, p)}$$

$$[\text{SR}]\ \frac{\forall o : \text{Object}^? \bullet (o, \text{rdf:type}, c) \in T \Rightarrow \exists s : \text{Subject}^? \bullet (s, p, o) \in T}{T \models (c, \text{mosto:strongRange}, p)}$$

7.8 Satisfaction of correspondences

Given an executable mapping, it is also important to know whether it satisfies a given correspondence or not. This prevents the executable mapping from constructing data that does not satisfy the correspondences in a data exchange problem.

In the next subsections, we describe how to infer whether an executable mapping satisfies a correspondence or not. We use symbol $\models$ to denote satisfaction, T_C to denote a set of triple patterns in a CONSTRUCT clause, T_W to denote a set of triple patterns in a WHERE clause, c, c_1, and c_2 to denote classes, p, p_1, and p_2 to denote data properties, and q_1 and q_2 to denote object properties.

7.8.1 Class to class correspondences

These correspondences specify that an instance of a class in the source has to be reclassified in the target. We formalise this idea as follows:

$$[\text{C2C}]\ \frac{\forall s : Subject^? \bullet (s, \texttt{rdf:type}, c_1) \in T_W \Rightarrow (s, \texttt{rdf:type}, c_2) \in T_C}{(T_C, T_W) \models (c_1, \texttt{mosto:classToClass}, c_2)}$$

7.8.2 Data property to data property correspondences

Such a correspondence specifies that, if there is an instance of a data property in the source, then there must be another instance of the corresponding data property in the target that must have the same object. The correspondence itself cannot specify what the subject is. This idea is formalised as follows:

$$[\text{D2D}]\ \frac{\begin{array}{c}\forall s : Subject^?; o : Object^? \bullet (s, p_1, o) \in T_W \Rightarrow \\ \exists s' : Subject^? \bullet (s', p_2, o) \in T_C\end{array}}{(T_C, T_W) \models (p_1, \texttt{mosto:dataToData}, p_2)}$$

7.8.3 Object property to object property correspondences

These correspondences specify that, for every instance of an object property in the source, there must exist another instance of the corresponding object property in the target. The correspondence itself cannot specify which the subjects and the objects are. This idea is described formally as follows:

$$[\text{O2O}]\ \frac{\begin{array}{c}\forall s : \text{Subject}^?; o : \text{Object}^? \bullet (s, q_1, o) \in T_W \Rightarrow \\ \exists s' : \text{Subject}^?; o' : \text{Object}^? \bullet (s', q_2, o') \in T_C\end{array}}{(T_C, T_W) \models (q_1, \text{mosto:objectToObject}, q_2)}$$

7.8.4 Data property to class correspondences

A correspondence of this kind specifies that the object of every instance of a data property in the source must be reclassified in the target. The following inference rule formalises this idea:

$$[\text{D2C}]\ \frac{\forall s : \text{Subject}^?; o : \text{Object}^? \bullet (s, p, o) \in T_W \Rightarrow (o, \text{rdf:type}, c) \in T_C}{(T_C, T_W) \models (p, \text{mosto:dataToClass}, c)}$$

7.9 Limitations

The conceptual framework described in this chapter has a number of limitations that make it unapplicable to some data exchange problems. In this section, we highlight these limitations.

Our conceptual framework is able to deal with six types of constraints and four types of correspondences. Regarding constraints, there are some OWL constructs with which our proposal cannot deal, namely: zero-cardinality restrictions, general property restrictions, and intersection of restrictions (see Appendix A for further details). Regarding correspondences, our conceptual framework can deal with class to class, data property to data property, object property to object property, and data property to class. However, some real-world data exchange problems might require to define other types of correspondences, such as class to data property, data property to object property, or object property to class.

Another limitation is that, in the constraints and correspondences that our conceptual framework defines, it is not possible to restrict more than two entities, and to establish that more than one source entity corresponds to a target entity. General OWL constructs allow to define constraints amongst an arbitrary number of entities, so our conceptual framework is not able to model these constraints. Furthermore, there are proposals that allow to define correspondences to combine a number of source entities that correspond to a single target entity using a transformation function, so our conceptual framework is not able to model these correspondences.

7.10 Summary

In this chapter, we have described the conceptual framework on which our proposal is based. It defined entities, literals, URIs, blank nodes, triples, ontologies, executable mappings, and data exchange problems. Furthermore, we have defined precisely what satisfying a constraint or a correspondence means. Finally, we have described the limitations of our conceptual framework.

Chapter 8

Automatic Generation of Executable Mappings

We're now on full automatic, in the hands of the computers.
Planet of the Apes (1968), Film

We describe MostoDE in this chapter. It is our proposal to automatically generate executable mappings in SPARQL by means of constraints and correspondences. Furthermore, we analyse MostoDE and present its limitations. The chapter is organised as follows: in Section 8.1, we introduce it; Section 8.2 describes our proposal; in Section 8.3, we analyse our proposal to prove that its worst-case complexity is computationally tractable, and we also prove that it is correct; Section 8.4 presents the limitations of our proposal; finally, we summarise the chapter in Section 8.5.

8.1 Introduction

In this chapter, we describe our proposal to automatically generate executable mappings in SPARQL. We rely on constraints, which relate entities of the source or the target of a data exchange problem, and correspondences, which relate a source entity with a target entity. First, our proposal computes the so-called kernels, which are data exchange subproblems that describe structures to be exchanged as a whole. Then, kernels are automatically transformed into executable mappings in SPARQL, which are used to perform data exchange.

In addition, we prove that our proposal is correct and computationally tractable, since $O(e_s^4\, e_t + e_s\, e_t^4 + e_s^2\, e_t^2(e_s + e_t))$ is an upper bound to its worst-time complexity, where e_s and e_t denote the number of entities in the source and target ontologies, respectively. Our proposal has a number of limitations that are described in this chapter, such as dealing with zero-cardinality restrictions, dealing with intersection of restrictions, or taking more than one instance of the same class or superclass into account.

8.2 Description of our proposal

Our proposal relies on the algorithm in Figure 8.1. It takes a data exchange problem as input and outputs a set of executable mappings. The algorithm loops through the set of correspondences and applies a series of steps to compute their kernels, initial executable mappings, variable links, and substitutions before creating the resulting executable mappings. The key feature of these mappings is that they retrieve data that satisfies the source constraints and constructs data that satisfies both the target constraints and the correspondences. At a first glance, it might be surprising that we take source constraints into account, since source data should ideally satisfy them; however, in the decentralised environment of the Semantic Web, it is common that real-world ontologies have data that do not satisfy their constraints [26]. The results of our experimental validation prove that this interpretation of correspondences seems to capture the intuition behind many common data exchange problems.

Below, we provide additional details on each of the steps of our algorithm. To illustrate them, we use a non-trivial, real-world data exchange problem that is based on the evolution from DBpedia 3.2 to DBpedia 3.6 [23], which involved many major structural changes. Figure 8.2 illustrates this problem using our graph-based notation (see Section 3.2).

```
 1: algorithm  generateExecutableMappings
 2: input      d : DataExchangeProblem
 3: output     M : ℙ ExecutableMapping
 4: variables  o1, o2 : Ontology; TC, TW : ℙ TriplePattern;
 5:            V : ℙ Correspondence; LW, LC, LV : ℙ Link; S : Substitution
 6:
 7: M := ∅
 8: for each v : Correspondence | v ∈ correspondences(d) do
 9:     -- Compute the kernel (o1, o2, V) of v
10:     o1 := findSubOntology(subject(v), source(d))
11:     o2 := findSubOntology(object(v), target(d))
12:     V := findCorrespondences(o1, o2, correspondences(d))
13:     -- Compute the initial executable mapping (TC, TW)
14:     TW := initialiseTriplePatterns(entities(o1))
15:     TC := initialiseTriplePatterns(entities(o2))
16:     -- Compute variable links in (TC, TW)
17:     LW := findConstraintLinks(TW, constraints(o1))
18:     LC := findConstraintLinks(TC, constraints(o2))
19:     LV := findCorrespondenceLinks(TW, TC, V)
20:     -- Compute substitutions and apply them to (TC, TW)
21:     S := findSubstitution(LW ∪ LC ∪ LV, TW)
22:     TC := applySubstitution(TC, S)
23:     TW := applySubstitution(TW, S)
24:     -- Update the resulting set of executable mappings
25:     M := M ∪ {(TC, TW)}
26: end for
```

Figure 8.1: *Algorithm to generate executable mappings.*

8.2.1 Step 1: Computing kernels

For every correspondence in the data exchange problem being analysed, our algorithm constructs a kernel, which is a data exchange subproblem. Intuitively, a kernel describes the structure of a subset of data in the source ontology that needs to be exchanged as a whole, and the structure of a subset of data in the target ontology that needs to be created as a whole; in other words, if more or less data are considered, then the exchange would be incoherent.

For instance, Figure 8.3 presents the kernel that is associated with the correspondence between property dbp32:academyaward and class dbp36:Award in our running example. This correspondence specifies that, for every instance

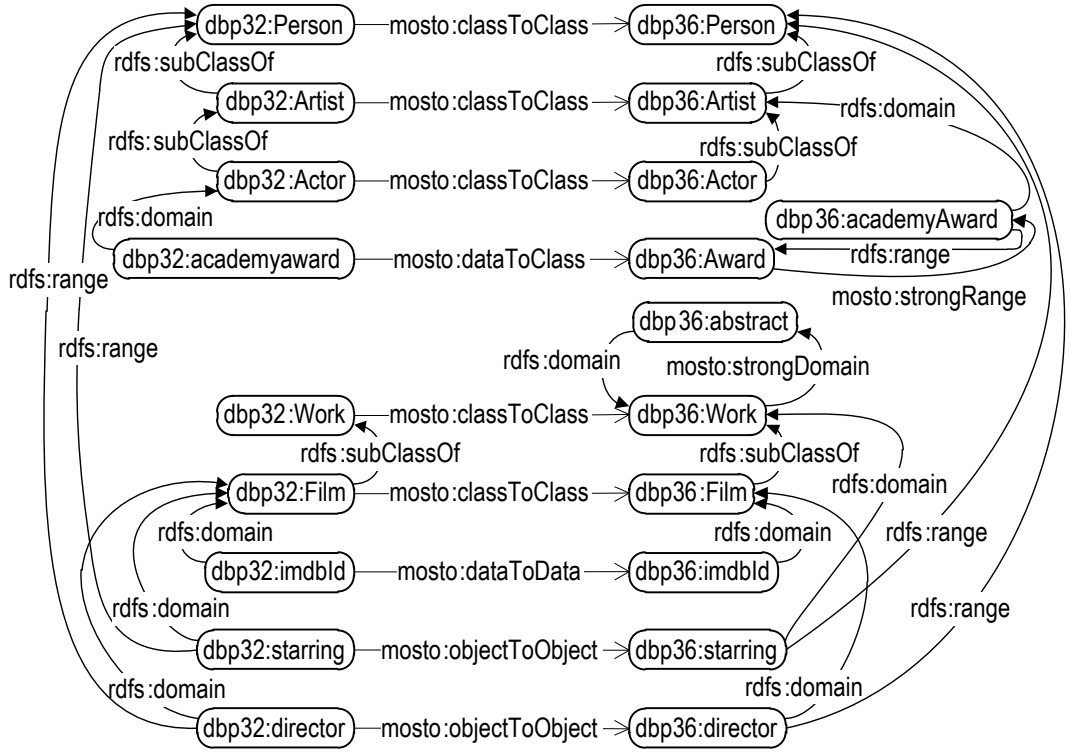


Figure 8.2: *A running example.*

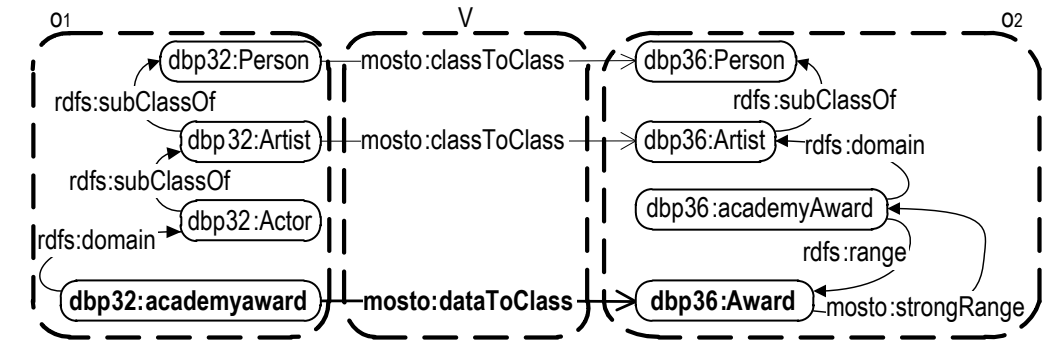


Figure 8.3: *A sample kernel.*

of property dbp32:academyaward that is found in the source ontology, the object of this instance must be classified as dbp36:Award in the target ontology. But property dbp32:academyaward has a domain constraint that relates it to class dbp32:Actor, which has a subclassing constraint that relates it to class dbp32:Artist, which has an additional subclassing constraint that relates it to class dbp32:Person. As a conclusion, the object of an instance of property dbp32:academyaward cannot be exchanged in isolation, but in the context of an instance of class dbp32:Actor that acts as the subject, which is also an instance of classes dbp32:Artist and dbp32:Person. Similarly, we cannot simply classify the object of property dbp32:academyaward as an instance of class dbp36:Award in the target data model, since this class is the strong range of property dbp36:academyAward, which in turn is related to class dbp36:Artist by means of domain constraints, and to dbp36:Person by means of a subclassing constraint. As a conclusion, neither can the instance of dbp36:Award be constructed in isolation, but in the context of an instance of property dbp36:academyAward that has an instance of classes dbp36:Artist and dbp36:Person as subject.

In the algorithm in Figure 8.1, the computation of the kernel that is associated with every correspondence is performed in lines 10–12. Given a correspondence that is referred to as v, we first find the subontologies that are associated with the subject and the object of v, and then find the correspondences between them.

We present the algorithm to compute subontologies in Figure 8.4. It works on an input entity e and an input ontology o, and returns an ontology (E, C) that results from exploring e in depth. The algorithm iterates over a set of entities Q that is initialised to $\{e\}$; intuitively, Q stores the entities that remain to be explored. In each iteration of the main loop, the algorithm removes an entity f from set Q and then calculates the subset of constraints whose subject is f, which is immediately added to the resulting set of constraints C; note, however, that we only add to Q the objects that have not been explored so far to prevent the algorithm from looping forever in the many common cases in which the ontology has cycles.

We present the algorithm to compute the correspondences between two subontologies in Figure 8.5. It takes two ontologies o_1 and o_2 and a set of correspondences V as input. The output is calculated as the subset of correspondences from V whose subject belongs to the entities of ontology o_1, and the object belongs to the set of entities of ontology o_2.

```
 1: algorithm  findSubOntology
 2: input      e : Entity; o : Ontology
 3: output     o′ : Ontology
 4: variables  E, Q : ℙ Entity; C, Z : ℙ Constraint; f : Entity
 5:
 6: E, C := ∅
 7: Q := {e}
 8: while Q ≠ ∅ do
 9:     f := pick an entity from Q
10:     Q := Q \ {f}
11:     E := E ∪ {f}
12:     Z := {z : Constraint | z ∈ constraints(o) ∧ subject(z) = f}
13:     C := C ∪ Z
14:     Q := Q ∪ {g : Entity | g ∉ E ∧ ∃z : Constraint • z ∈ Z ∧ object(z) = g}
15: end while
16: o′ := (E, C)
```

Figure 8.4: *Algorithm to find subontologies.*

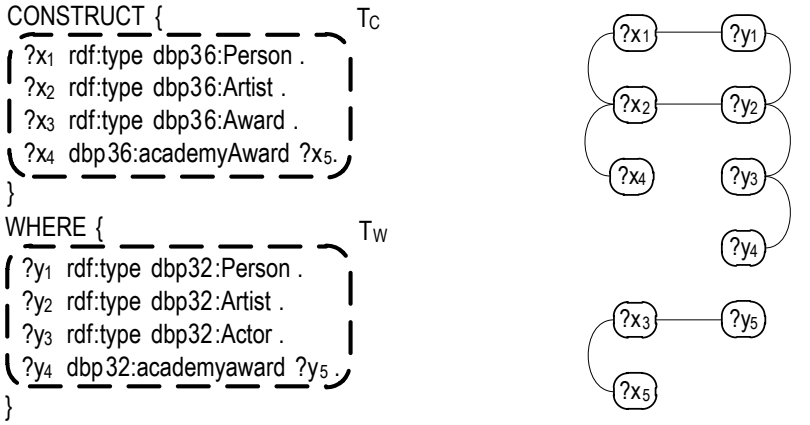
```
1: algorithm  findCorrespondences
2: input      o1, o2 : Ontology; V : ℙ Correspondence
3: output     V′ : ℙ Correspondence
4:
5: V′ = {v : Correspondence | v ∈ V ∧ subject(v) ∈ entities(o1) ∧
6:                            object(v) ∈ entities(o2)}
```

Figure 8.5: *Algorithm to find the correspondences between two subontologies.*

8.2.2 Step 2: Computing initial executable mappings

A kernel is just a starting point; it needs to be transformed into an executable mapping. The first step is to transform its source and its target subontologies into two sets of initial triple patterns. The set that corresponds to the WHERE clause must include a triple pattern to retrieve every instance of an entity in the source ontology of the kernel, whereas, the set that corresponds to the CONSTRUCT clause must include a triple pattern to construct an instance of every entity in the target ontology of the kernel.

Figure 8.6(a) presents the initial executable mapping of the kernel of Fig-



(a) *Initial executable mapping.*

(b) *Variable links.*

Figure 8.6: *Sample initial executable mapping and variable links.*

ure 8.3. For example, to retrieve the instances of class dbp32:Actor, we need a triple pattern of the form $(?y_3, \texttt{rdf:type}, \texttt{dbp32:Actor})$, where $?y_3$ denotes a fresh variable. Similarly, there is a property called dbp32:academyaward, which requires a triple pattern of the form $(?y_4, \texttt{dbp32:academyaward}, ?y_5)$, where $?y_4$ and $?y_5$ denote fresh variables, as well. Intuitively, we should link variables $?y_3$ and $?y_4$ since the subject of an instance of property dbp32:academyaward must be an instance of class dbp32:Actor; the initial executable mapping does not take these links into account, since computing them is the goal of the next step.

In the algorithm in Figure 8.1, the computation of the initial executable mapping is performed in lines 14 and 15. We present the algorithm to initialise the triple patterns in Figure 8.7. It takes a set of entities E as input and returns a set of triple patterns T. The resulting set is initialised to the empty set, and then the algorithm iterates over the set of entities. In each iteration, it adds a new triple pattern to the result set according to the type of entity being analysed, namely: for every class c, it adds a triple pattern of the form $(?x, \texttt{rdf:type}, c)$, where ?x denotes a fresh variable; and for every property p, it adds a triple pattern of the form $(?y, p, ?z)$, where ?y and ?z denote two fresh variables.

```
 1: algorithm  initialiseTriplePatterns
 2: input      E : ℙ Entity
 3: output     T : ℙ TriplePattern
 4:
 5: T := ∅
 6: for each e : Entity | e ∈ E do
 7:     if e ∈ Class then
 8:         T := T ∪ {(freshVar(), rdf:type, e)}
 9:     else if e ∈ Property then
10:         T := T ∪ {(freshVar(), e, freshVar())}
11:     end if
12: end for
```

Figure 8.7: *Algorithm to initialise triple patterns.*

8.2.3 Step 3: Computing variable links

The variables in the triple patterns generated by the previous step are pairwise distinct. This step is responsible for finding links amongst these variables. To find them, we need to analyse the constraints and the correspondences in the kernel that is associated with the correspondence being analysed.

Figure 8.6(b) presents the variable links regarding the initial executable mapping in Figure 8.6(a). Links can be naturally represented as an undirected graph in which the nodes are variables, and the edges indicate which variables are linked. For example, property dbp32:academyaward in our running example has a constraint of type rdfs:domain that indicates that its domain is class dbp32:Actor; this means that we need to locate the triple patterns that we generated for these entities and link their corresponding subjects, i.e., $?y_3$ and $?y_4$. Similarly, there exists a correspondence between property dbp32:academyaward and class dbp36:Award in our running example; this means that we need to locate the triple patterns that we generated for these entities and link $?y_5$ and $?x_3$.

We formally define the sets of links as follows:

$$Link == Variable \times Variable$$

In the algorithm in Figure 8.1, lines 17–19 compute variable links. We first find the links that are due to the constraints in the source ontology, then the

```
 1: algorithm  findConstraintLinks
 2: input      T : ℙ TriplePattern;  C : ℙ Constraint
 3: output     L : ℙ Link
 4: variables  t1, t2 : TriplePattern
 5:
 6: L := ∅
 7: for each z : Constraint | z ∈ C do
 8:     t1 := findTriplePattern(subject(z), T)
 9:     t2 := findTriplePattern(object(z), T)
10:     if predicate(z) ∈ {rdfs:subClassOf, rdfs:domain,
11:         mosto:strongDomain} then
12:         L := L ∪ {(subject(t1), subject(t2))}
13:     else if predicate(z) = rdfs:subPropertyOf then
14:         L := L ∪ {(subject(t1), subject(t2)), (object(t1), object(t2))}
15:     else if predicate(z) = rdfs:range then
16:         L := L ∪ {(object(t1), subject(t2))}
17:     else if predicate(z) = mosto:strongRange then
18:         L := L ∪ {(subject(t1), object(t2))}
19:     end if
20: end for
```

Figure 8.8: *Algorithm to find variable links using constraints.*

links that are due to the constraints in the target ontology, and then finish the process by finding the links that are due to the correspondences, themselves.

We present the algorithms to find the variable links due to constraints and correspondences in Figures 8.8 and 8.9, respectively. They both operate very similarly: they iterate over the set of input constraints or correspondences, find the triple patterns associated with their subjects and objects, and create the appropriate links. The only feature that requires a little explanation is that correspondences between object properties do not result in any links. The reason is that, according to inference rule O2O, an object property to object property correspondence does not specify how to link the subject or the object of the target instance; it only specifies that an instance of the source property must exist in the source triple patterns, and an instance of the target property must exist in the target triple patterns; these instances are generated in the second step.

The algorithm to find the triple pattern that is associated with an entity is presented in Figure 8.10. Due to the way we initialise triple patterns, their

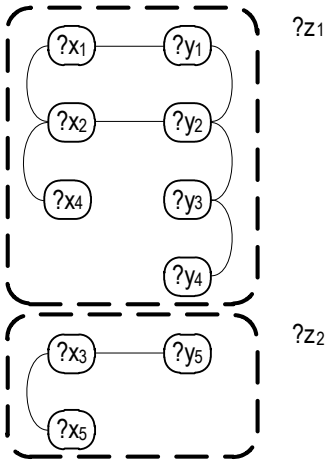
```
 1: algorithm  findCorrespondenceLinks
 2: input      T_W, T_C : ℙ TriplePattern;  V : ℙ Correspondence
 3: output     L : ℙ Link
 4: variables  t_1, t_2 : TriplePattern
 5:
 6: L := ∅
 7: for each v : Correspondence | v ∈ V do
 8:     t_1 := findTriplePattern(subject(v), T_W)
 9:     t_2 := findTriplePattern(object(v), T_C)
10:     if predicate(v) = mosto:classToClass then
11:         L := L ∪ {(subject(t_1), subject(t_2))}
12:     else if predicate(v) = mosto:dataToData then
13:         L := L ∪ {(object(t_1), object(t_2))}
14:     else if predicate(v) = mosto:dataToClass then
15:         L := L ∪ {(object(t_1), subject(t_2))}
16:     end if
17: end for
```

Figure 8.9: *Algorithm to find variable links using correspondences.*

```
 1: algorithm  findTriplePattern
 2: input      e : Entity;  T : ℙ TriplePattern
 3: output     t : TriplePattern
 4:
 5: t := nil
 6: for each u : TriplePattern | u ∈ T while t = nil do
 7:     if predicate(t) = e ∨ object(t) = e then
 8:         t := u
 9:     end if
10: end for
```

Figure 8.10: *Algorithm to find a triple pattern of an entity.*

subject is always a variable; only the predicate or the object can be entities. This is the reason why this algorithm does not check the subject of the triple patterns it examines.



(a) *Resulting substitution.*

CONSTRUCT { T_C
?z1 rdf:type dbp36:Person .
?z1 rdf:type dbp36:Artist .
?z2 rdf:type dbp36:Award .
?z1 dbp36:academyAward ?z2.
}
WHERE { T_W
?z1 rdf:type dbp32:Person .
?z1 rdf:type dbp32:Artist .
?z1 rdf:type dbp32:Actor .
?z1 dbp32:academyaward ?z2 .
}

(b) *Final executable mapping.*

Figure 8.11: *Sample substitution and executable mapping.*

8.2.4 Step 4: Computing and applying substitutions

The result of the previous step is a graph in which every connected component includes a group of variables that are linked, i.e., a group of variables that should actually be the same. Consequently, the next step is to transform this graph into a substitution in which every variable in every connected component is replaced for the same fresh variable, and then apply it to the initial executable mapping that we computed in the second step.

Figure 8.11(a) highlights the two connected components in the variable links in Figure 8.6(b); the substitution maps the variables in each connected component to fresh variables $?z_1$ and $?z_2$, respectively. Figure 8.11(b) shows the executable mapping that results from applying the previous substitution to the initial executable mapping in Figure 8.6(a).

In this example, every variable in the CONSTRUCT clause is linked to a variable in the WHERE clause. There can be, however, cases in which there exists a variable in the CONSTRUCT clause that is not linked to any variable in the WHERE clause. Whenever this happens, the interpretation is that the set of correspondences is not complete enough to describe the data exchange problem that is being analysed. In some situations, the set of correspondences can be completed to solve the problem, but there are others in which this is not possible because the target ontology provides more information than the

```
 1: algorithm  findSubstitution
 2: input      L : ℙ Link;  T_W : ℙ TriplePattern
 3: output     S : Substitution
 4: variables  CC : ℙ ℙ Link;  x : Variable ∪ BlankNode
 5:
 6: CC := findConnectedComponents(L)
 7: S := ∅
 8: for each K : ℙ Link | K ∈ CC do
 9:     if variables(K) ∩ variables(T_W) ≠ ∅ then
10:         x := freshVar()
11:     else
12:         x := freshBlankNode()
13:     end if
14:     S := S ∪ {y, z : Variable | (y, z) ∈ K • y ↦ x} ∪
15:                 {y, z : Variable | (y, z) ∈ K • z ↦ x}
16: end for
```

Figure 8.12: *Algorithm to find substitutions.*

source ontology. For instance, DBPedia 3.6 provides information about work abstracts (property dbp36:abstract), which is not present in DBpedia 3.2. In these cases, it makes sense to generate a blank node that acts as a placeholder; in other words, instead of failing to exchange any data due to this problem, we can exchange as much data as possible and highlight special cases using blank nodes. These placeholders are known as labelled nulls in the context of nested-relational data models [37].

We formally define a substitution as a finite map from variables onto variables and blank nodes, namely:

$$Substitution == Variable \twoheadrightarrow Variable \cup BlankNode$$

In the algorithm in Figure 8.1, the computation and the application of substitutions are performed in lines 21–23. We first find the substitution that corresponds to the variable links that we have found in the third step, and then apply it to both the initial set of triple patterns in the CONSTRUCT and the WHERE clauses that we calculated in the second step.

We present the algorithm to find substitutions in Figure 8.12. It takes a set of links L and a set of triple patterns T_W as input and returns a substitution S; we implicitly assume that T_W is the set of triples in the WHERE clause of an executable mapping. The algorithm first invokes findConnectedComponents

```
 1: algorithm  applySubstitution
 2: input      T : ℙ TriplePattern; S : Substitution
 3: output     T′ : ℙ TriplePattern
 4: variables  s : Subject?; o : Object?
 5:
 6: T′ := ∅
 7: for each t : TriplePattern | t ∈ T do
 8:     s := subject(t)
 9:     o := object(t)
10:     if s ∈ dom S then
11:         s := S(s)
12:     end if
13:     if o ∈ dom S then
14:         o := S(o)
15:     end if
16:     T′ := T′ ∪ {(s, predicate(t), o)}
17: end for
```

Figure 8.13: *Algorithm to apply substitutions.*

to find the set of connected components CC in the input variable links; we do not provide any additional details on this algorithm since it is well-known in the literature [51]. It then initialises S to the empty set and iterates through the set of connected components CC. In each iteration, it checks if a component K includes variables from both the CONSTRUCT and the WHERE clauses, in which case a fresh variable is created; otherwise, we have found a group of variables for which there is not a correspondence that assigns values to them, which justifies the creation of a fresh blank node. Immediately after, it updates the resulting substitution S by mapping every pair of variables to the fresh variable or blank node that was created previously.

We present the algorithm to apply a substitution in Figure 8.13. It takes a substitution S and a set of patterns T as input and returns a new set of patterns T′ that results from applying substitution S to every subject and object in T.

8.3 Analysis of our proposal

In this section, we analyse our proposal to prove that its worst-case complexity is computationally tractable, and we also prove that it is correct.

8.3.1 Analysis of complexity

In the following theorems and propositions, we analyse the worst-case complexity of our algorithm and its subalgorithms. The worst case is a data exchange problem in which every entity of the source ontology has a correspondence with every entity of the target ontology. Furthermore, the source and target ontologies are complete graphs, i.e., every pair of entities are connected by a constraint. As a conclusion, in the worst case problem, $v = e_s\, e_t$, $c_s = (e_s^2 - e_s)/2$, and $c_t = (e_t^2 - e_t)/2$, where e_s and e_t denote the number of entities in the source and target ontologies, v denotes the number of correspondences, and c_s and c_t denote the number of source and target constraints.

In our proofs, we assume that simple set operations like invoking a projection function, checking for membership, merging two sets, or constructing a tuple can be implemented in $O(1)$ time. We also implicitly assume that data exchange problems must be finite, i.e., the sets of entities, constraints, and correspondences involved are finite.

Theorem 8.1 (Generating executable mappings (see Figure 8.1)) *Let* d *be a data exchange problem.* $O(e_s^4\, e_t + e_s\, e_t^4 + e_s^2\, e_t^2(e_s + e_t))$ *is an upper bound for the worst-time complexity of Algorithm* `generateExecutableMappings(d)`*, where* e_s *and* e_t *denote the number of entities in the source and target of* d*, respectively.*

Proof Algorithm `generateExecutableMappings(d)` has to iterate through the whole set of correspondences in d. It calls `findSubOntology` two times for each correspondence (lines 10 and 11): the first time is to compute the source subontology, and the second time to compute the target subontology, which, according to Proposition 8.1, terminate in $O(e_s\, c_s)$ and $O(e_t\, c_t)$ time, respectively, where $c_s = (e_s^2 - e_s)/2$ is the number of source constraints, and $c_t = (e_t^2 - e_t)/2$ is the number of target constraints. In the next step, the algorithm calls `findCorrespondences` (line 12), which, according to Proposition 8.2, terminates in $O(v)$ time, where $v = e_s\, e_t$ denotes the correspondences of d. Furthermore, the algorithm calls `initialiseTriplePatterns` two times (lines 14 and 15): the first time is to compute the initial source triple patterns and the second time to compute the initial target triple patterns, which, according to Proposition 8.3, terminates in $O(e_s)$ and $O(e_t)$ time, respectively. In the following steps, the algorithm calls `findConstraintLinks` two times (lines 17 and 18): the first time is to compute the links that are related to the source constraints, and the second time to compute the links that are related to the target constraints, which, according to Proposition 8.4, terminate in $O(c_s\, t_w)$ and $O(c_t\, t_c)$ time, where $c_s = (e_s^2 - e_s)/2$ is the number

of source constraints, $t_w = e_s$ is the number of triple patterns in the WHERE clause, which is equal to the whole set of source entities in the worst case, $c_t = (e_t^2 - e_t)/2$ is the number of target constraints, and $t_c = e_t$ is the number of triple patterns in the CONSTRUCT clause, which is equal to the whole set of target entities in the worst case. In the next step, the algorithm calls `findCorrespondenceLinks` (line 19), which, according to Proposition 8.5, terminates in $O(v\,(t_w + t_c))$ time, where $v = e_s\,e_t$ denotes the correspondences of d, $t_w = e_s$ is the number of triple patterns in the WHERE clause, and $t_c = e_t$ is the number of triple patterns in the CONSTRUCT clause. In the next step, the algorithm calls `findSubstitution` (line 21), which, according to Proposition 8.7, terminates in $O(l)$ time, where $l = 2\,c_s + 2\,c_t + v = e_s^2 - e_s + e_t^2 - e_t + e_s\,e_t$ is the total number of links which, in the worst case, comprise two links for each source and target constraint (subproperty constraint), and a link for each correspondence. Finally, the algorithm calls `applySubstitution` two times (lines 22 and 23): the first time is to compute the substitution of the WHERE clause, and the second time to compute the substitution of the CONSTRUCT clause, which, according to Proposition 8.8, terminate in $O(t_w)$ and $O(t_c)$ time, respectively, where $t_w = e_s$ is the number of triple patterns in the WHERE clause, and $t_c = e_t$ is the number of triple patterns in the CONSTRUCT clause.

Therefore, we get the following expression: $O(e_s\,e_t(e_s/2\,(e_s^2 - e_s) + e_t/2\,(e_t^2 - e_t) + e_s\,e_t + e_s + e_t + e_s/2\,(e_s^2 - e_s) + e_t/2\,(e_t^2 - e_t) + e_s\,e_t(e_s + e_t) + e_s^2 - e_s + e_t^2 - e_t + e_s\,e_t + e_s + e_t)) = O(e_s^4\,e_t + e_s\,e_t^4 + 2\,e_s^2\,e_t^2 + e_s^3\,e_t^2 + e_s^2\,e_t^3 + e_s^2\,e_t + e_s\,e_t^2)$, in which $e_s^4\,e_t > e_s^2\,e_t$, $e_s\,e_t^4 > e_s\,e_t^2$, and $e_s^2\,e_t^3 > 2\,e_s^2\,e_t^2$. As a conclusion, $O(e_s^4\,e_t + e_s\,e_t^4 + e_s^2\,e_t^2(e_s + e_t))$ is an upper bound for the worst-time complexity of Algorithm `generateExecutableMappings(d)`. □

Proposition 8.1 (Finding subontologies (see Figure 8.4)) *Let* f *be an entity and* o *an ontology.* `findSubOntology(f, o)` *terminates in* $O(e\,c)$ *time in the worst case, where* e *and* c *denote the number of entities and constraints in ontology* o*, respectively.*

Proof In the worst case, `findSubOntology(f, o)` has to iterate through the whole set of entities in o; additionally, in each iteration, it has to iterate through the whole set of constraints. As a conclusion, `findSubOntology(f, o)` terminates in $O(e\,c)$ time in the worst case. □

Proposition 8.2 (Finding correspondences (see Figure 8.5)) *Let* o_1 *and* o_2 *be two ontologies, and* V *a set of correspondences. In the worst case,* `findCorrespondences(o1, o2, V)` *terminates in* $O(v)$ *time, where* v *denotes the number of correspondences in set* V*.*

Proof Algorithm findCorrespondences has to iterate through the whole set of correspondences V to find the ones whose subject belongs in o_1 and whose object belongs in o_2. As a conclusion, findCorrespondences(o_1, o_2, V) terminates in $O(v)$ time in the worst case. □

Proposition 8.3 (Initialising triple patterns (see Figure 8.7)) *Let* E *be a set of entities.* initialiseTriplePatterns(E) *terminates in* $O(e)$ *time in the worst case, where* e *denotes the number of entities in set* E*.*

Proof Algorithm initialiseTriplePatterns has to iterate through the whole set of entities E. As a conclusion, initialiseTriplePatterns(E) terminates in $O(e)$ time in the worst case. □

Proposition 8.4 (Finding constraint links (see Figure 8.8)) *Let* T *be a set of triple patterns, and* C *a set of constraints,* findConstraintLinks(T, C) *terminates in* $O(c\,t)$ *time in the worst case, where* t *denotes the number of triple patterns in* T *and* c *denotes the number of constraints in* C*.*

Proof Algorithm findConstraintLinks iterates over the whole set of input constraints C. In each iteration, it invokes findTriplePattern on T twice, and each invocation terminates in $O(t)$ time in the worst case according to Proposition 8.6. As a conclusion, findConstraints(T, C) terminates in $O(c\,t)$ time in the worst case. □

Proposition 8.5 (Finding correspondence links (see Figure 8.9)) *Let* T_W *and* T_C *be two sets of triple patterns, and* V *an arbitrary set of correspondences.* findCorrespondenceLinks(T_W, T_C, V) *terminates in* $O(v\,(t_w + t_c))$ *time in the worst case, where* v *denotes the number of correspondences in* V*, and* t_w *and* t_c *denote the number of triple patterns in* T_W *and* T_C*, respectively.*

Proof Algorithm findCorrespondenceLinks iterates through the whole set of input correspondences V. In each iteration, it invokes Algorithm findTriplePattern on both T_W and T_C; these invocations terminate in $O(t_w + t_c)$ time in the worst case according to Proposition 8.6. As a conclusion, Algorithm findCorrespondenceLinks terminates in $O(v\,(t_w + t_c))$ time in the worst case. □

Proposition 8.6 (Finding triple patterns (see Figure 8.10)) *Let* e *be an entity, and* T *a set of triple patterns.* findTriplePattern(e, T) *terminates in* $O(t)$ *time in the worst case, where* t *denotes the number of triple patterns in* T*.*

Proof Algorithm findTriplePattern iterates through the whole set of triple patterns T, as long as it does not find the triple pattern that refers to *e* in its predicate or its object. In the worst case, this triple is the last one. As a conclusion, findTriplePattern, terminates in $O(t)$ time in the worst case. □

Proposition 8.7 (Finding substitutions (see Figure 8.12)) *Let* L *be a set of variable links, and* T_W *a set of triple patterns.* $O(l)$ *is an upper bound for the worst-time complexity of* $\texttt{findSubstitution}(L, T_W)$*, where* l *denotes the number of links in* L*.*

Proof Algorithm `findConnectedComponents` terminates in $O(\max\{a, l\})$ time in the worst case [51], where a denotes the number of variables in L. The algorithm then iterates through the set of connected components in L; it is not easy to characterise the worst case, but it is safe to assume that l must be an upper bound to the number of connected components that is returned by `findConnectedComponents` since a graph with l edges may have a maximum of l connected components. Therefore, $O(\max\{a, l\} + l)$ is an upper bound to the worst-case time complexity of Algorithm `findSubstitution`. If $\max\{a, l\} = a$, then the upper bound is $O(a + l)$; contrarily, if $\max\{a, l\} = l$, then the upper bound is $O(l)$. Therefore, $O(a + l)$ is also an upper bound for the worst-case time complexity of this algorithm. Note that, in the worst case, $a = 2l$ since all variables related by the links are pairwise distinct. As a conclusion, $O(l)$ is an upper bound to the worst-case time complexity of Algorithm `findSubstitution`. □

Proposition 8.8 (Applying substitutions (see Figure 8.13)) *Let* T *be a set of triple patterns and* S *a substitution.* $\texttt{applySubstitution}(T, S)$ *terminates in* $O(t)$ *time in the worst case, where* t *denotes the cardinality of* T*.*

Proof Algorithm `applySubstitution` has to iterate through the whole set of triple patterns T. As a conclusion, `applySubstitution` terminates in $O(t)$ time in the worst case. □

8.3.2 Analysis of correctness

In the following theorem and propositions, we prove that our algorithm is correct. This requires us to prove that the executable mappings it generates retrieve source data that satisfy the source constraints and exchanges them into target data that satisfy the target constraints and the correspondences.

Theorem 8.2 (Generating executable mappings) *Let* d *be a data exchange problem.* $\texttt{generateExecutableMappings}(d)$ *outputs a number of executable mappings that satisfy the following properties: i) every source triple it retrieves satisfies the constraints of* $\texttt{source}(d)$*; ii) every target triple it generates satisfies the constraints of* $\texttt{target}(d)$*; and, iii) they satisfy the correspondences in the kernels from which they originate.*

Proof The proof builds on Propositions 8.9, 8.10, and 8.11, in which we prove each property of the resulting executable mappings independently. □

Proposition 8.9 (Source constraint satisfaction) *Let* d *be a data exchange problem and* $m = (T_C, T_W)$ *any of the executable mappings that are returned by* `generateExecutableMappings`(d). m *retrieves data that satisfies the constraints of the source ontology of* d.

Proof The proof follows from reductio ad absurdum: assume that z is a constraint in the source of d, and that T_W does not satisfy it. Depending on the predicate of z, we may distinguish the following cases:

- If constraint z is of the form $(c_1, \texttt{rdfs:subClassOf}, c_2)$, where c_1 and c_2 denote two classes, then the initial triple patterns are of the form $(?x_1, \texttt{rdf:type}, c_1)$ and $(?x_2, \texttt{rdf:type}, c_2)$; thus, the algorithm to find constraint links must return a link between variables $?x_1$ and $?x_2$, which, after computing the corresponding substitution and applying it to the initial triple patterns, results in two triple patterns of the form $(?x, \texttt{rdf:type}, c_1)$ and $(?x, \texttt{rdf:type}, c_2)$. According to inference rule SC, T_W satisfies constraint z.

- If z is of the form $(p_1, \texttt{rdfs:subPropertyOf}, p_2)$, where p_1 and p_2 denote two properties, then the initial triple patterns are of the form $(?x_1, p_1, ?y_1)$ and $(?x_2, p_2, ?y_2)$; thus, the algorithm to find constraint links must return a link between variables $?x_1$ and $?x_2$, and another link between variables $?y_1$ and $?y_2$, which, after computing the corresponding substitution and applying it to the initial triple patterns, result in two triple patterns of the form $(?x, p_1, ?y)$ and $(?x, p_2, ?y)$. According to inference rule SP, T_W satisfies constraint z.

- If z is of the form $(p, \texttt{rdfs:domain}, c)$, where p denotes a property and c denotes a class, then the initial triple patterns are of the form $(?x_1, p, ?y_1)$ and $(?x_2, \texttt{rdf:type}, c)$; thus the algorithm to find constraint links must return a link between variables $?x_1$ and $?x_2$, which, after computing the corresponding substitution and applying it to the initial triple patterns, results in two triple patterns of the form $(?x, p, ?y_1)$ and $(?x, \texttt{rdf:type}, c)$. According to inference rule D, T_W satisfies constraint z.

- If z is of the form $(p, \texttt{rdfs:range}, c)$, where p denotes a property and c a class, then the initial triple patterns are of the form $(?x_1, p, ?y_1)$ and $(?x_2, \texttt{rdf:type}, c)$; thus the algorithm to find constraint links must return

a link between variables $?y_1$ and $?x_2$, which, after computing the corresponding substitution and applying it to the initial triple patterns, results in two triple patterns of the form $(?x_1, p, ?y)$ and $(?y, \texttt{rdf:type}, c)$. According to inference rule R, T_W satisfies constraint z.

- If z is of the form $(c, \texttt{mosto:strongDomain}, p)$, where c denotes a class and p a property, then the initial triple patterns are of the form $(?x_1, \texttt{rdf:type}, c)$ and $(?x_2, p, ?y_2)$; thus the algorithm to find constraint links must return a link between variables $?x_1$ and $?x_2$, which, after computing the corresponding substitution and applying it to the initial triple patterns, results in two triple patterns of the form $(?x, \texttt{rdf:type}, c)$ and $(?x, p, ?y_2)$. According to inference rule SD, T_W satisfies constraint z.

- If z is of the form $(c, \texttt{mosto:strongRange}, p)$, where c denotes a class and p a property, then the initial triple patterns are of the form $(?x_1, \texttt{rdf:type}, c)$ and $(?x_2, p, ?y_2)$; thus the algorithm to find constraint links must return a link between variables $?x_1$ and $?y_2$, which, after computing the corresponding substitution and applying it to the initial triple patterns, results in two triple patterns of the form $(?y, \texttt{rdf:type}, c)$ and $(?x_2, p, ?y)$. According to inference rule SR, T_W satisfies constraint z.

Since we have found a contradiction in every case, we can conclude that the initial hypothesis is wrong. As a conclusion, m retrieves data that satisfies the constraints of the source ontology of d. □

Proposition 8.10 (Target constraint satisfaction) *Let* d *be a data exchange problem and* $m = (T_C, T_W)$ *any of the executable mappings that are returned by* $\texttt{generateExecutableMappings}(d)$. m *constructs data that satisfies the constraints of the target ontology of* d.

Proof The proof follows straightforwardly using the same reasoning as in the previous proposition. □

Proposition 8.11 (Correspondence satisfaction) *Let* d *be a data exchange problem and* $m = (T_C, T_W)$ *any of the executable mappings that are returned by* $\texttt{generateExecutableMappings}(d)$. *Assume that* v *is the correspondence from which* $\texttt{generateExecutableMappings}$ *generated* m*, and that* k *is the kernel associated with* v. m *satisfies the subset of correspondences in* k.

Proof The proof follows from reductio ad absurdum: assume that v is a correspondence in $\texttt{correspondences}(k)$, and that m does not satisfy it. Depending on the predicate of v, we may distinguish the following cases:

- If v is of the form $(c_1, \text{mosto:classToClass}, c_2)$, where c_1 and c_2 denote two classes, the initial triple patterns of the WHERE and the CONSTRUCT clauses contain two triple patterns of the form $(?x_1, \text{rdf:type}, c_1)$ and $(?y_1, \text{rdf:type}, c_2)$, respectively; the algorithm to find correspondence links must then find a link between variables $?x_1$ and $?y_1$, which, after computing the corresponding substitution and applying it, results in triple patterns $(?x, \text{rdf:type}, c_1)$ and $(?x, \text{rdf:type}, c_2)$. According to inference rule C2C, m satisfies this correspondence.

- If v is of the form $(p_1, \text{mosto:dataToData}, p_2)$, where p_1 and p_2 denote two data properties, the initial triple patterns of the WHERE and the CONSTRUCT clauses contain two triple patterns of the form $(?x_1, p_1, ?y_1)$ and $(?x_1, p_2, ?y_2)$, respectively; the algorithm to find correspondence links must then find a link between variables $?y_1$ and $?y_2$, which, after computing the corresponding substitution and applying it, results in triple patterns $(?x_1, p_1, ?y)$ and $(?x_2, p_2, ?y)$. According to inference rule D2D, m satisfies this correspondence.

- If v is of the form $(p_1, \text{mosto:objectToObject}, p_2)$, where p_1 and p_2 denote two object properties, the initial triple patterns of the WHERE and the CONSTRUCT clauses contain two triple patterns of the form $(?x_1, p_1, ?y_1)$ and $(?x_1, p_2, ?y_2)$, respectively; the algorithm to find correspondence links ignores this kind of correspondences and that, according to inference rule O2O, the initial triple patterns satisfy this correspondence.

- If v is of the form $(p, \text{mosto:dataToClass}, c)$, where p denotes a data property and c denotes a class, the initial triple patterns of the WHERE and the CONSTRUCT clauses contain two triple patterns of the form $(?x_1, p, ?y_1)$ and $(?x_2, \text{rdf:type}, c)$, respectively; the algorithm to find correspondence links must then find a link between variables $?y_1$ and $?x_2$, which, after computing the corresponding substitution and applying it, results in triple patterns $(?x_1, p, ?y)$ and $(?y, \text{rdf:type}, c)$. According to inference rule D2C, m satisfies this correspondence.

Since we have found a contradiction in every case, we can conclude that the initial hypothesis is wrong. As a conclusion, m satisfies the correspondences in the kernel that is associated with the correspondence from which m originated. □

8.4 Limitations

Our proposal has a number of limitations that make it unapplicable to some data exchange problems. In this section, we highlight these limitations.

Our proposal relies on a conceptual framework in which it is not possible to restrict more than two entities, and to establish that more than one source entity correspond to a target entity (see Section 7.9). Regarding constraints, if this type of constraint were necessary, it is mandatory to change the algorithm to find subontologies in kernels (see Section 8.2.1). Regarding correspondences, in our tool that implements this proposal [95], we allow to define correspondences from one or more source entities, which may be also related by a function to exchange them, into a single target entity.

An additional limitation is that our proposal cannot deal with more than one instance of the same class or superclass (except rdfs:Resource and owl:Thing). For instance, if a kernel comprises both dbp32:Person and dbp32:Author classes, the final executable mapping shall comprise a single variable ?x of both types, i.e., $(?x, \texttt{rdf:type}, \texttt{dbp32:Person})$ and $(?x, \texttt{rdf:type}, \texttt{dbp32:Author})$. Therefore, it is not possible to have two or more variables of type dbp32:Person or dbp32:Author. Two problems are derived from this fact: the first one is that, if an ontology comprises a class that is a superclass of the rest of the classes, the exchange produces incoherent target data; the second problem is that properties that have the same class as domain and range cannot be appropriately exchanged. For instance, property sch:marriedTo has the same domain and range, i.e., class dbp32:Person, and the generated executable mappings in this case only exchanges a person that is married to herself or himself, which does not make sense.

Our proposal does not generate SPARQL executable mappings that include triple patterns with regular expressions [5]. This implies that we are not able to deal with RDF collections, such as bags, lists, or sequences [27]. Furthermore, it may generate equivalent SPARQL executable mappings, i.e., executable mappings that exchange exactly the same source data into the same target data. Note that it is not trivial to detect this type of executable mappings, so we do not try to remove them. This implies that the exchange of data may be inefficient because the same data is exchanged more than once, but it does not generate incorrect target data.

8.5 Summary

This chapter has described our proposal on the automatic generation of executable mappings in SPARQL. It relies on constraints and correspondences

amongst source and target ontologies. We have proved that our proposal is computationally tractable and correct, and we have presented its limitations.

Chapter 9

Benchmarking Semantic Data Exchange Systems

Like a diet of the mind, I just choose not to indulge certain appetites; like my appetite for patterns; perhaps my appetite to imagine and to dream.

A Beautiful Mind (2001), Film

In this chapter, we present MostoBM, a benchmark for testing semantic data exchange systems in the context of ontologies and query engines. The chapter is organised as follows: in Section 9.1, we introduce it; Section 9.2 describes the real-world data exchange patterns included in our benchmark; in Section 9.3, we present the synthetic data exchange patterns that our benchmark comprises; Section 9.4 presents a number of parameters that allow to scale the data exchange patterns of our benchmark; finally, we summarise the chapter in Section 9.5.

9.1 Introduction

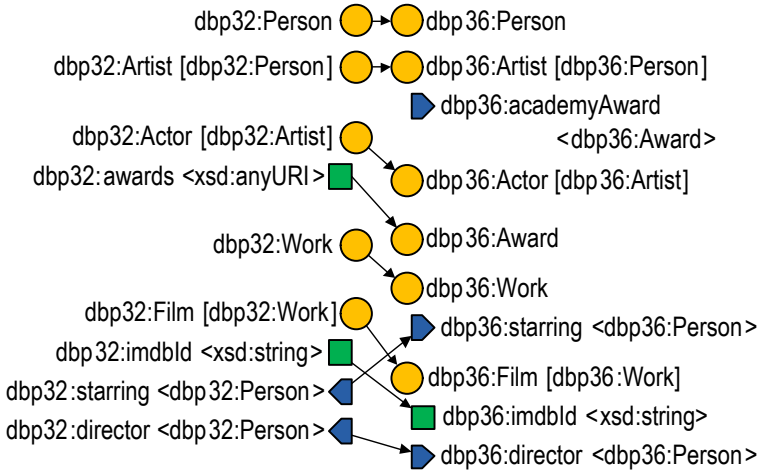
MostoBM is a benchmark for testing semantic data exchange systems in the context of ontologies and query engines. Our benchmark provides a catalogue of three real-world and seven synthetic data exchange patterns; seven parameters to construct scenarios that are instantiations of the patterns; and a publicly available tool that facilitates the instantiation of the patterns, and the gathering of data about the performance of systems [70].

The three real-world patterns are relevant data exchange problems in the context of Linked Open Data, whereas the seven synthetic ones are common integration problems that are based on current proposals in the ontology evolution context and on our experience regarding real-world information integration problems. This catalogue is not meant to be exhaustive: the patterns described in this dissertation are the starting point to a community effort that is expected to extend them.

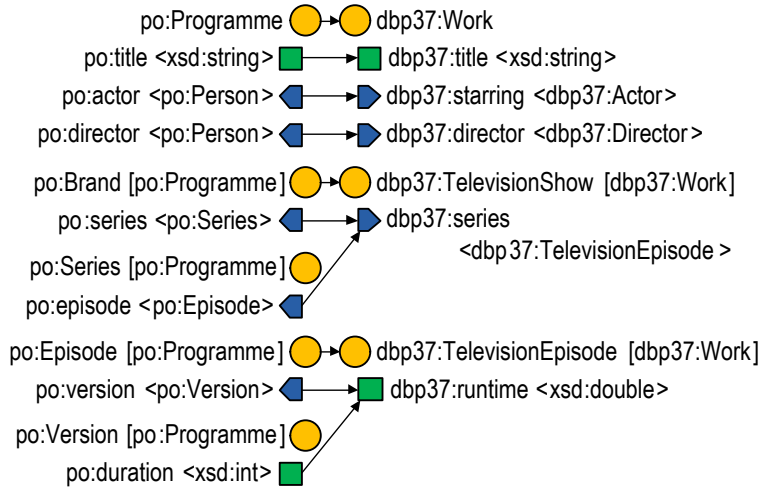
A benchmark should be scalable and the results that it produces should be deterministic and reproducible [43]. To fulfill these properties, MostoBM provides a number of parameters to construct and/or populate scenarios, each of which is a three-element tuple (S, T, Q), where S is the source ontology, T is the target ontology, and Q is a set of SPARQL queries to perform data exchange. The parameters allow to scale the data of the source ontology for the real-world patterns, in which the structure of the source and target ontologies and the SPARQL queries are fixed. Furthermore, the parameters allow to scale the structure of source and target ontologies, the data of the source ontology, and the SPARQL queries to perform data exchange for the synthetic patterns. Thanks to them, we can automatically construct the structure of a source and a target ontology with, for instance, a thousands classes, a dozen specialisation or object property levels, or the data of a source ontology with a million triples. The scaling of the patterns helps analyse the performance of semantic data exchange systems in future, when it is assumed that data exchange problems are going to increase their scale in structure and/or data.

9.2 Real-world patterns

Our benchmark provides three real-world data exchange patterns, each of which is instantiated into a variety of scenarios using a number of parameters (see Section 9.4). These patterns are illustrated in Figure 9.1 and presented below. The source ontology is on the left side and the target is on the right side; the arrows represent correspondences between the entities of the ontologies.



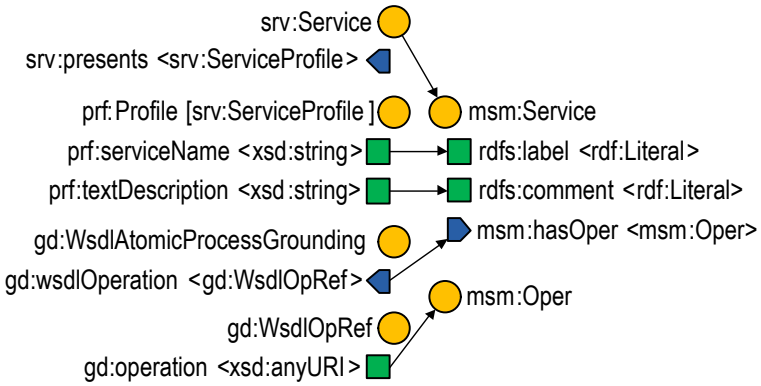
(a) *Evolving DBpedia.*



(b) *Adapting BBC Programmes to DBpedia.*

Figure 9.1: *Real-world patterns of our benchmark.*

These correspondences are visual hints to help readers understand each real-world pattern [19]. We selected these three patterns to be part of our benchmark because they represent integration problems that are common in practice in the context of Linked Open Data. We use some prefixes to denote different ontologies, such as dbp32:, dbp36:, *srv*: or po:.



(c) *Publishing OWL-S services as Linked Open Data.*

Figure 9.1: *Real-world patterns of our benchmark (Cont'd).*

9.2.1 Evolution of an ontology

Usually, ontologies change in response to a certain need [38], including that the domain of interest has changed, the perspective under which the domain is viewed needs to be changed, or due to design flaws in the original ontology. In this context, the source ontology is the ontology before changes are applied and the target ontology is the ontology after changes are applied.

This pattern focuses on DBpedia [23], which comprises a number of different versions due to a number of changes in its conceptualisation. When a new version of DBpedia is devised, the new ontology may be populated by performing data exchange from a previous version to the new one. In this pattern, we perform data exchange from a part of DBpedia 3.2 that focuses on artists, actors, directors, and films to DBpedia 3.6 (see Figure 9.1(a)).

9.2.2 Vocabulary adaptation

It is not uncommon that two ontologies offer the same data structured according to different vocabularies. Therefore, we wish to adapt the vocabulary of a source ontology to the vocabulary of a target ontology.

This pattern focuses on the BBC Programmes and DBpedia ontologies. The BBC [56] decided to adhere to the Linked Data principles a couple of years ago. They provide ontologies that adhere to the these principles to publicise

the music and programmes they broadcast in both radio and television. In this pattern, which is shown in Figure 9.1(b), we perform data exchange from the Programmes Ontology 2009, which describes programmes including brands, series (seasons), and episodes, to a part of DBpedia 3.7 that models television shows and episodes.

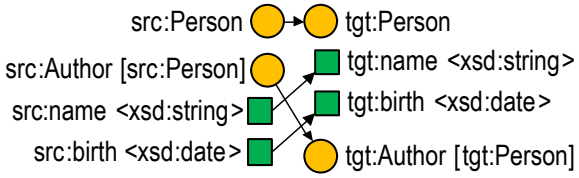
9.2.3 Publication of Linked Open Data

Before the success of the Linked Open Data initiative, there were a variety of ontologies that publish their data without taking Linked Data principles into account. Usually, there is a need to transform these ontologies into ontologies that publish their data using these principles.

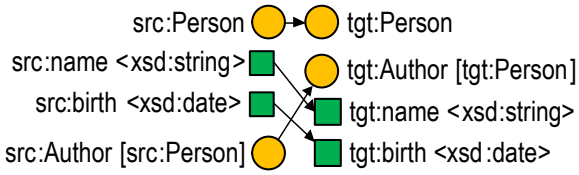
This pattern focuses on publishing semantic web services as Linked Open Data. OWL-S [54] is one of the main proposals for describing semantic web services that defines an upper ontology in OWL. MSM (Minimal Service Model) [81] is a web service ontology that allows to publish web services as Linked Open Data. Furthermore, it provides a lightweight solution to integrate service descriptions based on different ontologies, such as OWL-S, WSMO, and others. In this pattern, which is shown in Figure 9.1(c), we publish OWL-S 1.1 services as Linked Open Data by means of MSM 1.0.

9.3 Synthetic patterns

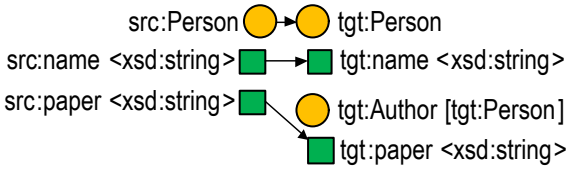
A synthetic data exchange pattern represents a common and relevant integration problem. Our benchmark provides a catalogue of seven synthetic data exchange patterns; to design them, we have leveraged our experience on current proposals in the ontology evolution context (and on our experience regarding real-world information integration problems in the context of ontology evolution (DBpedia 3.2 and DBpedia 3.6), multimedia content (Programmes Ontology 2009 and DBpedia 3.7), and semantic web services (OWL-S 1.1 and MSM 1.0). Each pattern represents an intention of change [38], i.e., each pattern represents a number of atomic changes that are applied to an ontology in response to certain needs. In addition, each synthetic pattern is instantiated into a variety of scenarios using a number of parameters (see Section 9.4). Below, we present our synthetic patterns, which are illustrated in Figure 9.2. Note that `src:` and `tgt:` prefixes are used for the source and target ontologies, respectively.



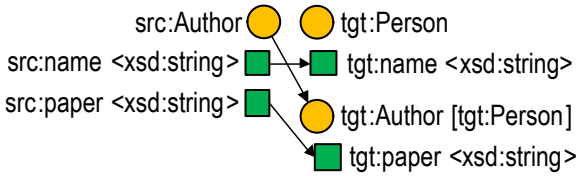
(a) *Lift Properties.*



(b) *Sink Properties.*



(c) *Extract Subclasses.*



(d) *Extract Superclasses.*

Figure 9.2: *Synthetic patterns of our benchmark.*

9.3.1 Lift Properties

The intention of change is that the user wishes to extract common properties to a superclass in a taxonomy. Therefore, the data properties of a set of subclasses are moved to a common superclass. In the example, src:name and src:birth data properties are lifted to tgt:name and tgt:birth, respectively.

9.3.2 Sink Properties

The intention of change is that the user wishes to narrow the domain of a number of properties. Therefore, the data properties of a superclass are moved

to a number of subclasses. In the example, src:name and src:birth data properties are sunk to tgt:name and tgt:birth, respectively.

9.3.3 Extract Subclasses

The intention of change is that the user wishes to specialise a class. Therefore, a source class is split into several subclasses and the domain of target data properties is selected amongst the subclasses. In the example, every instance of src:Person is transformed into an instance of tgt:Person. After performing data exchange in this example, all target instances are of type tgt:Person. However, if the knowledge is made explicit, every instance of tgt:Person, which is related to a data property instance of tgt:paper, is also an instance of type tgt:Author.

9.3.4 Extract Superclasses

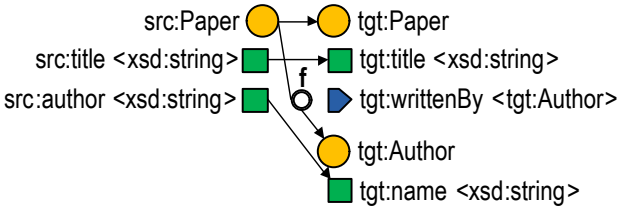
The intention of change is that the user wishes to generalise a class. Therefore, a class is split into several superclasses, and data properties are distributed amongst them. After performing data exchange in this example, all target instances are of type tgt:Author, which are implicitly instances of tgt:Person, too.

9.3.5 Extract Related Classes

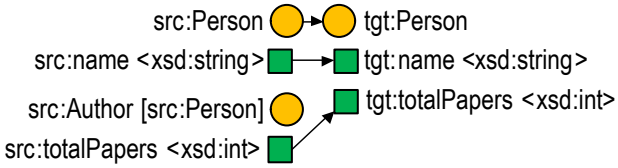
The intention of change is that the user wishes to extract a number of classes building on a single class. Therefore, the data properties that have this single class as domain change their domains by the new classes, which are related to the original one by a number of object properties. In the example, the source class src:Paper is split into two target classes called tgt:Paper and tgt:Author that are related by object property tgt:writtenBy. Instances of tgt:Author are constructed by applying a user-defined function f to the instances of src:Paper.

9.3.6 Simplify Specialisation

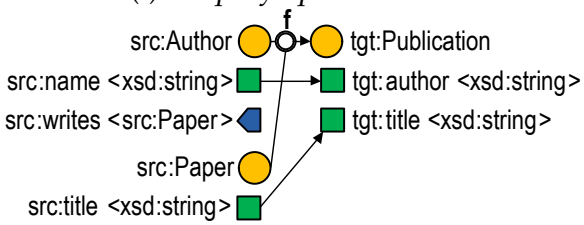
The intention of change is that the user wishes to remove a taxonomy of classes. Thus, a set of specialised classes are flattened into a single class. In the example, src:Person and src:Author, which is an specialisation of src:Person, are simplified to tgt:Person.



(e) *Extract Related Classes.*



(f) *Simplify Specialisation.*



(g) *Simplify Related Classes.*

Figure 9.2: *Synthetic patterns of our benchmark (Cont'd).*

9.3.7 Simplify Related Classes

The intention of change is that the user wishes to join a set of classes that are related by object properties. Therefore, several source classes are transformed into one class that aggregates them all. In the example, for every two instances of src:Author and src:Paper related by src:writes, a new instance of tgt:Publication is constructed.

9.4 Parameters

Our benchmark takes a number of input parameters that allow to tune the data of the source ontology in the real-world patterns, and both structure and data of the source and/or the target ontologies in the synthetic patterns. Thanks to them, a user is able to instantiate a scenario of a pattern.

The structure parameters are the following:

- Levels of classes ($L \in \mathbb{N}$): number of relationships (specialisations or object properties) amongst one class and the rest of the classes in the source or target ontologies. L allows to scale the structure of ontologies in depth.
- Number of related classes ($C \in \mathbb{N}$): number of classes related to each class by specialisation or object properties. C allows to scale the structure of ontologies in width.
- Number of data properties ($D \in \mathbb{N}$): of the source and target ontologies.

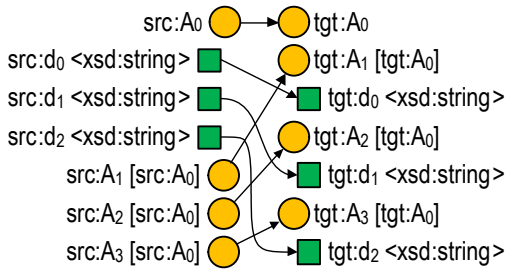
Note that L and C may be applied to both source and target ontologies, which is the case of the Lift Properties and Sink Properties patterns; to the target ontology only, i.e., the Extract Subclasses, Extract Superclasses and Extract Related Classes patterns; or to the source ontology only, i.e., the Simplify Specialisation and Simplify Related Classes patterns. The total number of classes an ontology comprises is computed by the following formula: $\sum_{i=0}^{L} C^i$.

Figure 9.3(a) shows a sample instantiation of the Sink Properties pattern in which $L = 1$, $C = 3$, $D = 3$. Structure parameters also have an effect on the SPARQL queries constructed by our benchmark, since they vary depending on the structure of the source and target ontologies. Figure 9.3(b) shows two sample queries for the sample scenario of the Sink Properties pattern: Q_1 is responsible for reclassifying A_0 in the source as A_1 in the target, and Q_2 is responsible for both reclassifying A_0 as A_1 in the target, and exchanging the value of $\texttt{src:}d_0$ into $\texttt{tgt:}d_0$.

Structure parameters can be tuned to construct realistic ontologies, e.g., ontologies in the context of life sciences usually comprise a large set of classes that form a wide and/or deep taxonomy. For instance, the Gene Ontology [13] comprises roughly $32,000$ classes with a dozen specialisation levels. Therefore, to construct ontologies that resemble the Gene Ontology, we need to specify the following parameters $L = 14$, $C = 2$.

Regarding data parameters, they are used to scale the instances of the source ontology, and these parameters are the following:

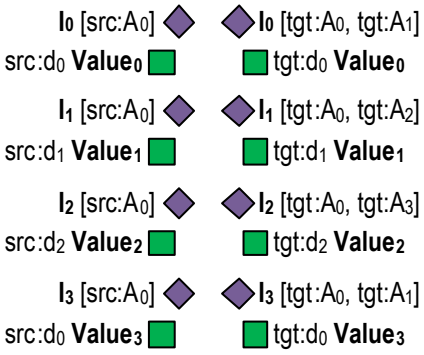
- Number of individuals ($I \in \mathbb{N} \setminus \{0\}$): number of instances of owl:Thing that the source ontology shall have.
- Number of types ($I_T \in \mathbb{N}$): number of types that each individual shall have.



(a) *Structure.*

```
Q1: CONSTRUCT {
  ?a rdf:type tgt:A0.
} WHERE {
  ?a rdf:type src:A0. }
Q2: CONSTRUCT {
  ?a rdf:type tgt:A1.
  ?a tgt:d0   ?d.
} WHERE {
  ?a rdf:type src:A0.
  ?a src:d0   ?d. }
```

(b) *Queries.*



(c) *Data.*

Figure 9.3: *Sample scenario of the Sink Properties pattern.*

- Number of data properties ($I_D \in \mathbb{N}$): number of data property instances for which a given individual is the subject.
- Number of object properties ($I_O \in \mathbb{N}$): number of object property instances for which a given individual is the subject.

The actual types and instances may be randomly selected from the whole set of classes, data properties and object properties of the ontology to be populated. In our tool, we provide 44 statistical distributions to randomly select them, including Uniform, Normal, Exponential, Zipf, Pareto and empirical distributions, to mention a few. As a result, it is possible to use statistical distributions that model real-world ontologies.

Figure 9.3(c) shows an example of the data constructed to populate the source ontology (left side) in Figure 9.3(a), in which the data parameters are

the following: $I = 4$, $I_T = 1$, $I_D = 1$, $I_O = 0$. Furthermore, this figure shows the target instances (right side) that result from performing data exchange with the SPARQL queries in Figure 9.3(b). We may compute the number of data triples that a populated ontology comprises by means of the following formula: $I\ (1 + I_T + I_D + I_O)$.

Data parameters can be tuned to populate ontologies that resemble realistic ones, e.g., the SwetoDBLP [1] ontology models computer science publications. It comprises roughly two million triples of individuals of a single type, four million triples of data property instances, and seven million triples of object property instances. To simulate this ontology, we have to set the following values: $I = 2 \times 10^6$, $I_T = 1$ (the individuals are of a single type), $I_D = 2$ (four million triples of data property instances divided by two million triples of individuals), and $I_O = 4$ (seven million triples of object properties divided by two million triples of individuals).

9.5 Summary

In this chapter, we have presented our benchmark to test semantic data exchange systems in the context of ontologies and query engines, which is called MostoBM. We have introduced the real-world and synthetic data exchange patterns that our benchmark comprises. Furthermore, it provides a number of parameters that allow to scale the previously described data exchange patterns.

Chapter 10

Experimental Evaluation

I have devised seven separate explanations, each of which would cover the facts as far as we know them. But which of these is correct can only be determined by the fresh information which we shall no doubt find waiting for us.

Sir Arthur Conan Doyle, The Adventure of the Copper Beeches (1892)

We validate our proposal by checking if our interpretation of correspondences agrees with the interpretation of domain experts. Furthermore, we provide an evaluation methodology that allows to compare semantic data exchange systems side by side. The chapter is organised as follows: in Section 10.1, we introduce it; Section 10.2 describes the validation of MostoDE using our benchmark; Section 10.3 presents an evaluation methodology to make informed and statistically-sound decisions regarding semantic data exchange systems; Sections 10.4 and 10.5 present examples of how to apply the methodology to a number of real-world and synthetic data exchange patterns; finally, we summarise the chapter in Section 10.6.

10.1 Introduction

Executable mappings encodes how correspondences between source and target data models must be interpreted, i.e., how to perform data exchange. Correspondences are inherently ambiguous since there can be many different executable mappings that satisfy them, but generate different target data. Therefore, it is mandatory to validate if the interpretation that our proposal assumes is coherent with expected results by domain experts.

To perform this validation, we have devised a repository that comprises a number of real-world and synthetic data exchange problems. The goal of this repository is to provide a number of data exchange problems that should be supported by every proposal that deals with SPARQL executable mappings.

In addition, we provide an evaluation methodology that allows to compare semantic data exchange systems side by side. To the best of our knowledge, this is the first such evaluation methodology in the literature. This methodology helps software engineers make informed and statistically-sound decisions based on rankings that focus on: 1) which semantic data exchange system performs better; and 2) how the performance of systems is influenced by the parameters of MostoBM. This methodology is benchmark-agnostic, i.e., it can be applied to any real-world or synthetic pattern, and technology-agnostic, i.e., it can be applied to any system.

10.2 Validation of our proposal

Proving correctness implies that our proposal does not generate data that do not satisfy the source constraints, the target constraints, or the correspondences. Since an executable mapping encodes an interpretation of the correspondences in a data exchange problem, it is then necessary to check if our interpretation of correspondences agrees with the interpretation of domain experts.

Unfortunately, there does not exist a standard repository of data exchange problems on which different proposals can be tested and compared. To address this problem, we have set up a repository using MostoBM. This repository comprises four real-world data exchange problems, which have been constructed using the three real-world data exchange patterns of our benchmark (see Section 9.2) and a fictitious ontology of movies (see below), and $3,780$ synthetic problems, which have been constructed using the seven synthetic data exchange patterns of our benchmark (see Section 9.3).

	DBP	O2M	MO	LP (540)	SP (540)
Classes	12	72	9	[4 - 84]	[4 - 84]
Data properties	4	41	8	[50 - 154]	[50 - 154]
Object properties	5	90	8	0	0
Correspondences	9	11	10	[27 - 115]	[27 - 115]
Source constraints	12	696	54	[26 - 489]	[26 - 489]
Target constraints	49	118	58	[26 - 489]	[26 - 489]
Triples	2,107,451	2,536,567	1,093,928	[871 - 18,564]	[769 - 14,183]
Executable mappings	9	11	10	[27 - 115]	[27 - 115]
Precision	1.0	1.0	1.0	1.0	1.0
Recall	1.0	1.0	1.0	1.0	1.0
Generation time (secs)	0.06	0.25	0.08	[0.02 - 0.11]	[0.03 - 0.1]
Data exchange (secs)	2.95	55.2	20.62	[0.14 - 0.69]	[0.14 - 0.72]

Table 10.1: *Summary of our validation.*

Regarding the real-world data exchange problems, we included the following: (DBP) exchanging data from DBpedia 3.2 to DBPedia 3.6, (BBC) exchanging data from Programmes Ontology 2009 to DBpedia 3.7, (O2M) exchanging data from OWL-S 1.1 to MSM 1.0, and (MO) exchanging data from both DBpedia 3.6 and Review 2.0 into a fictitious Movies On-line ontology.

The synthetic problems included 540 problems of each of the following synthetic data exchange patterns (see Section 9.3): (LP) Lifting properties, (SP) Sinking properties, (ESB) Extracting subclasses, (ESP) Extracting superclasses, (ERC) Extracting related classes, (SS) Simplifying specialisations, and (SRC) Simplifying related classes.

We used our tool to create a set of executable mappings for each data exchange problem in the repository, and then ran them and compared the results with the expected ones. The experiments were run on a computer that was equipped with a single 2.66 GHz Core 2 Duo CPU and 4 GB RAM, Windows XP Professional (SP3), JRE 1.6.0, Jena 2.6.4, and Oracle 11.2.0.1.0.

Table 10.1 summarises our results. The columns represent the data exchange problems, and the rows a number of measures; the first group of measures provides an overall idea of the size of each data exchange problem, whereas the second group provides information about the number of executable mappings, the precision an recall we attained when we executed them, the time to generate them, and the time they took to execute, i.e., the time of

	ESB (540)	ESP (540)	ERC (540)	SS (540)	SRC (540)
Classes	[3 - 45]	[3 - 45]	[3 - 45]	[3 - 45]	[3 - 45]
Data properties	[50 - 154]	[50 - 154]	[50 - 154]	[50 - 154]	[50 - 154]
Object properties	0	0	[1 - 43]	0	[1 - 43]
Correspondences	[26 - 76]	[26 - 76]	[27 - 115]	[26 - 76]	[26 - 76]
Source constraints	[25 - 309]	[25 - 309]	[25 - 309]	[26 - 489]	[29 - 660]
Target constraints	[26 - 489]	[51 - 564]	[29 - 660]	[25 - 309]	[25 - 309]
Triples	[776 - 8,717]	[776 - 8,717]	[776 - 8,717]	[872 - 18,591]	[1355 - 17,400]
Executable mappings	[26 - 76]	[26 - 76]	[27 - 115]	[26 - 76]	[26 - 76]
Precision	1.0	1.0	1.0	1.0	1.0
Recall	1.0	1.0	1.0	1.0	1.0
Generation time (secs)	[0.02 - 0.06]	[0.02 - 0.15]	[0.02 - 0.34]	[0.02 - 0.09]	[0.02 - 0.34]
Data exchange (secs)	[0.14 - 0.63]	[0.14 - 0.63]	[0.17 - 2.42]	[0.14 - 0.44]	[0.16 - 8.31]

Table 10.1: *Summary of our validation (Cont'd).*

performing the data exchange. In the case of synthetic problems, we provide intervals that indicate the minimum and maximum value of each measure.

Our proposal is not able to deal with the BBC problem, since the Programmes Ontology 2009 comprises a single class that is superclass of all the classes of the ontology and this is a limitation of our proposal (see Section 8.4). Furthermore, our proposal achieved 100% precision and recall in all of our experiments, which reveals that the interpretation of the correspondences that we encode in our executable mappings captures the intuition behind them.

In the DBP problem, it is worth noting that BDpedia 3.6 provides more data than DBpedia 3.2; in this case, recall and precision were measured on the subset of DBpedia 3.6 that can be exchanged from DBpedia 3.2, since the remaining data resulted, obviously, in blank nodes. Note, too, that the time our proposal took to generate the mappings was less than one second in all cases; since timings are imprecise in nature, we repeated each experiment 25 times and averaged the results after discarding roughly 0.01% outliers using the well-known Chevischev's inequality. We also measured the time our executable mappings took to execute. Although these timings depend largely on the technology being used, i.e., the database used to persist triples and the SPARQL engine used to query them, we think that presenting them is appealing insofar they prove that the queries we generate can be executed on reasonably-large ontologies in a sensible time. In our experiments, we used Jena 2.6.4 as the database and ARQ 2.8.8 as the query engine.

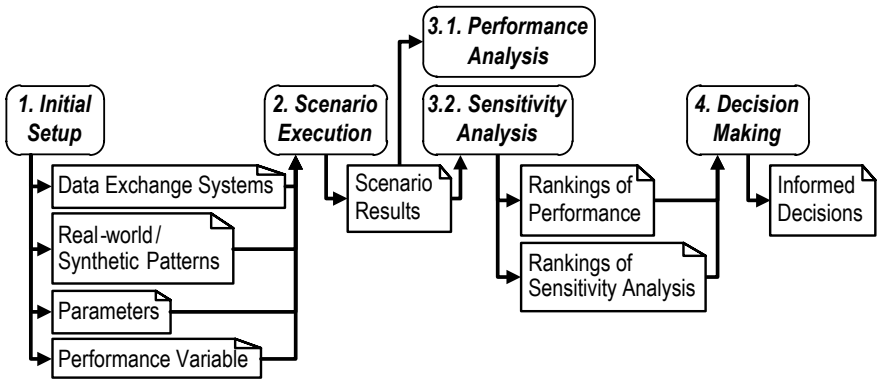


Figure 10.1: *Workflow of the evaluation methodology.*

10.3 Evaluation Methodology

Having a catalogue of data exchange patterns is not enough to evaluate a semantic data exchange system in practice. It is necessary to rely on a disciplined evaluation methodology if we wish to make informed and statistically-sound decisions. We have devised such a methodology and present its workflow in Figure 10.1.

We refer to performing data exchange over a scenario as executing that scenario. In the Initial Setup step, the user is responsible for selecting the semantic data exchange systems to test, the patterns to test these systems, the values of parameters, and a variable to measure the performance of these systems. The Scenario Execution step consists of executing scenarios of each pattern using each system. After executing the scenarios, the results are used to compute which system performs better (Performance Analysis step) and to analyse the influence of parameters in their performance (Sensitivity Analysis step). Finally, in the Decision Making step, the user is responsible for making decisions based on the previous analysis. In the rest of this section, we describe each of these steps in detail.

1. Initial Setup First, we need to select the study we are going to conduct. In this initial setup we are responsible for selecting the semantic data exchange systems to test: $\mathbb{M} = \{m_1, \ldots, m_a\}$, $a \geq 2$; the data exchange patterns: $\mathbb{P} = \{p_1, \ldots, p_b\}$, $b \geq 1$; the set of values for the parameters, and a performance variable. Our methodology is devised to focus only on real-world or synthetic patterns, but not both at the same time, since they entail the study

of different sets of parameters: in the real-world patterns, we only study data parameters since the structure of the source and target ontologies is fixed; in the synthetic patterns, we study both the structure and data parameters. Regarding structure parameters, we refer to their sets as P_L, P_C, P_D; regarding the data parameters, we refer to their sets as P_I, P_{IT}, P_{ID}, P_{IO}; in both cases, the subindex refers to the corresponding parameter.

A configuration is a tuple that comprises a value for each parameter, we denote the set of all possible configurations as: $\mathbb{C}$. When dealing with real-world patterns, $\mathbb{C} = P_I \times P_{IT} \times P_{ID} \times P_{IO}$; when dealing with synthetic patterns, $\mathbb{C} = P_L \times P_C \times P_D \times P_I \times P_{IT} \times P_{ID} \times P_{IO}$. The combination of a pattern and a configuration forms a scenario, we denote the set of all possible scenarios as: $\mathbb{X} = \mathbb{P} \times \mathbb{C}$. In addition, a setting is a combination of a system and a pattern, we denote the set of all possible settings as: $\mathbb{S} = \mathbb{M} \times \mathbb{P}$. Consequently, the total number of scenarios for each pattern is $|\mathbb{C}|$.

Regarding the performance variable, there are two types of variables that we can select to measure performance of systems: context-sensitive or context-insensitive. On the one hand, context-sensitive variables are affected by other processes that are executed in parallel in the computer, such as antiviruses, automatic updates, or backup processes. Therefore, it is mandatory to execute the scenarios a number of times, usually 25–30 times, and compute the average of the performance variable, removing possible outliers. Some examples of these performance variables are the following: user time, I/O time, or transferred bytes. On the other hand, context-insensitive variables are not affected by other processes, so it is not needed to execute the scenarios more than once; some examples of these performance variables are the following: CPU time, number of target triples, or memory used.

2. Scenario Execution For each system $m \in \mathbb{M}$ and each pattern $p \in \mathbb{P}$, we need to execute the scenarios related to p using m, so we need to execute the scenarios of all possible settings. Typical setups may involve the execution a large number of scenarios, each of which may take hours or even days to complete; this makes it necessary to use the Monte Carlo method to select a subset of scenarios to execute randomly. The problem that remains is to determine which the size of this subset must be.

Our evaluation methodology comprises a sensitivity analysis for which we use a regression method. This imposes an additional constraint on the number of scenarios to execute, since it is known that the ratio of scenarios to parameters must be at least 10 [67]. As a conclusion, when dealing with real-world patterns, we should execute at least 40 scenarios, since they involve four data

parameters, and when dealing with synthetic patterns, we should execute at least 70 scenarios, since they involve three structure and four data parameters. To provide a more precise figure on the number of scenarios to execute, we can use an iterative method that relies on Cochran's formula [31], which is based on the variance of this variable.

In this iterative method, we introduce a threshold to limit the total number of scenarios to execute: $\mu \in \mathbb{R}, 0 < \mu \leq 1$, since, in the worst case, this number is equal to $|\mathbb{C}|$, which means that the variance being studied is so high that it is not possible to calculate an average at confidence level 95% with an estimated error less than 3% (the standard values of Cochran's formula). The same occurs with the number of iterations of this method, i.e., a large number of iterations of this method means that the variance being studied is high; therefore, we introduce another threshold regarding the number of iterations: $\delta \in \mathbb{N}, \delta > 0$. We describe the iterative method below:

i. For each setting s_i, we select 40 or 70 scenarios to execute using the Monte Carlo method.

ii. We execute the selected scenarios.

iii. We use Cochran's formula to compute the new number of scenarios to execute using the variance of the performance variable, which is computed from the scenarios previously executed.

iv. Let e be the new number of scenarios to be executed, let k be the current number of iterations, and w the number of scenarios already executed, we have the following possibilities: 1) $e < w$ and $\delta \leq k$: we stop executing more scenarios and continue with the next step of the methodology; 2) $\mu|\mathbb{C}| > e > w$ and $\delta \leq k$: we select $e - w$ scenarios using the Monte Carlo method and return to the second step of this method; 3) $e > \mu|\mathbb{C}|$ or $\delta > k$: we discard s_i since it is not possible to make informed and statistically-sound decisions about this setting.

At the end of this step, we have results of the performance variable for each setting, i.e., a set of tuples of the form (s, c, v), where s is a setting, c is a configuration, and v the corresponding value of the performance variable.

3.1. Performance Analysis To compute which system performs better, we need a method to compare the values of the performance variable we gathered in the previous step. We can consider them as samples of an unknown random variable, which implies that we need to rely on statistical inference.

We apply Kruskal-Wallis's H test [50] to the results of each pattern in isolation, and to all of the results. The goal of this test is to determine if there are statistically significant differences amongst the performance variable for the systems under test. If the result is that there is no difference amongst the systems, it means that they all behave statistically identically with respect to the performance variable.

If the result of Kruskal-Wallis's H test is that there are statistically significant differences amongst the performance variable for the systems, we then use Wilcoxon's Signed Rank test to compute a rank amongst these systems [50]. This test outputs a p-value for each pair of systems, but it should not be interpreted as usual, but using Bonferroni's correction [68]. According to this correction, the p-value must not be compared with α, the confidence level, usually 0.05, but α/β, where β denotes the number of parameters, i.e., $\beta = 4$ when dealing with real-world patterns, and $\beta = 7$ when dealing with synthetic patterns. In other words, given systems m_1 and m_2, if Wilcoxon's Signed Rank test returns a p-value that is less than α/β, then the conclusion is that there is not enough evidence in our experiments to reject the hypothesis that m_1 performs better than m_2 regarding the performance variable being studied.

As a conclusion, the result of this step is a number of rankings of systems for each individual pattern and for all of them, i.e., a set of tuples of the form (P, r), in which $P \subseteq \mathbb{P}$ is a subset of patterns, and r is a ranking of systems.

3.2. Sensitivity Analysis In this step, we need a method to analyse the influence of parameters in the performance of systems. For each setting s, we use ReliefF to rank the influence of the parameters on the performance variable [99]. ReliefF is a general parameter estimator based on regression, which detects conditional dependencies amongst parameters and the performance variable. As a result, it outputs a number of numerical coefficients for each parameter, a lower value of these coefficients means that the parameter influences less on the performance variable.

Before using ReliefF, we need to normalise the values of the performance variable that we have measured in our scenarios, since ReliefF coefficients are not comparable. To perform this normalisation, we take the minimum and maximum values of the performance variable in the execution of scenarios for each setting, v_m and v_M, respectively, and we use the following formula: $v'_i = (v_i - v_m)/v_M$, where v_i is the value of the performance variable in each scenario and v'_i is the normalised value.

After applying ReliefF to our results, we obtain a ranking of the influence of parameters on the performance variable for each setting. Then, it is possible to rank the influence of parameters by clustering systems and/or patterns. Both types of rankings are computed by means of the Majoritarian Compromise method [105], which transforms a number of individual rankings into a single global ranking. To rank the influence by system, we take the individual rankings of each system and use the Majoritarian Compromise method to compute the global ranking. Similarly, to rank the influence by pattern, we compute the global ranking taking the individual rankings of each pattern into account.

At the end of this step, we have a number of rankings on the influence of parameters for each setting, for each system and every pattern, and for each pattern and every system, i.e., tuples of the form (m, p, r_x), (m, P, r_y) and (M, p, r_z), respectively, where m denotes a system, p denotes a pattern, $P \subseteq \mathbb{P}$ denotes a subset of patterns, $M \subseteq \mathbb{M}$ denotes a subset of systems, and r_x, r_y and r_z denote rankings on the influence of parameters.

4. Decision Making In this step, the user can make an informed and statistically-sound decision on which the best system is regarding the patterns and the performance variable that have been analysed.

10.4 Example with real-world patterns

To perform this example of our evaluation methodology, we used a tool implemented using Java that allows to execute the scenarios on several semantic data exchange systems. The tool was run on a virtual computer that was equipped with a four-threaded Intel Xeon 3.00 GHz CPU and 16 GB RAM, running on Windows Server 2008 (64-bits), JRE 1.6.0, Oracle 11.2.0.1.0, Jena 2.6.4, ARQ 2.8.7, TDB 0.8.7, Pellet 2.2.2, and OWLIM 4.2. Regarding the execution of scenarios, between each scenario execution, we forced our benchmark to wait for ten seconds to ensure that the Java garbage collector had finished collecting old objects from memory. Furthermore, we dropped and created the Oracle's tablespace in which we store the ontologies in each scenario execution.

1. Initial Setup For this example, we selected ten semantic data exchange systems to test (recall that a system comprises an RDF store, a reasoner, and a query engine). The selected systems were the following, namely: m_1 = Jena & ARQ & Jena Reasoner; m_2 = Jena & ARQ & Pellet; m_3 = Jena & ARQ & Oracle

Pattern	p_1			
System	Size	Ratio	Iter.	Accept
m_1	3908	39,08%	1	×
m_2	4939	49,39%	1	×
m_3	64	0,00%	2	✓
m_4	276	0,03%	4	✓
m_5	4957	49,57%	1	×
m_6	184	0,02%	3	✓
m_7	584	0,06%	2	✓
m_8	4964	49,64%	1	×
m_9	50	0,00%	2	✓
m_{10}	2174	21,74%	4	✓

Table 10.2: *Results of the iterative method (real-world patterns).*

Patterns	Rankings
$\{p_1\}$	$m_{10} > m_7 > m_4 > m_6 > m_3 > m_9$

Table 10.3: *Results of the performance analysis (real-world patterns).*

Reasoner; m_4 = Jena & ARQ & OWLIM Reasoner; m_5 = TDB & ARQ & Pellet; m_6 = TDB & ARQ & Oracle Reasoner; m_7 = TDB & ARQ & OWLIM Reasoner; m_8 = Oracle & Oracle & Pellet; m_9 = Oracle & Oracle & Oracle Reasoner; m_{10} = Oracle & Oracle & OWLIM Reasoner. Furthermore, we selected one real-world pattern to test them: p_1 = Publication of Linked Open Data. We also selected the following values: $P_I = \{1, 250, 500, 750, 1000\}$; $P_{IT} = P_{ID} = P_{IO} = \{1, 2, 3, 4, 5\}$, which amounts to $10,000$ scenarios for the pattern. Finally, we selected CPU time as the performance variable.

2. Scenario Execution We selected the following thresholds: $\mu = 0.25$, $\delta = 5$. Then, for each system and pattern, we used our iterative method to execute the scenarios. Table 10.2 shows our results for each system and pattern; the first column indicates the number of scenarios to execute according to Cochran's formula, the second column shows the ratio between the scenarios that we should execute and the total number of scenarios, the third column indicates

Systems	Patterns	Rankings
{m_3}	{p_1}	$I > I_T > I_O > I_D$
{m_4}	{p_1}	$I > I_T > I_D > I_O$
{m_6}	{p_1}	$I > I_T > I_O > I_D$
{m_7}	{p_1}	$I > I_T > I_O > I_D$
{m_9}	{p_1}	$I > I_T > I_O > I_D$
{m_{10}}	{p_1}	$I > I_T > I_O > I_D$
{$m_3, m_4, m_6, m_7, m_9, m_{10}$}	{p_1}	$I > I_T > I_O > I_D$

Table 10.4: *Results of the sensitivity analysis (real-world patterns).*

the number of iterations using our method, and the fourth column indicates if we accept or discard a particular system.

Taking these results into account, we discarded a semantic data exchange system if the ratio were greater than 25% ($\mu = 0.25$), or the iterations of the method were greater than five ($\delta = 5$). Therefore, we discarded the following systems: m_1, m_2, m_5, and m_8. Note also that, in Table 10.2, the ratio between executed scenarios and parameters for the remaining systems does not fall below ten, i.e., all of them are greater than 40.

3.1. Performance Analysis We took the result times of the previous step, and we first used Kruskal-Wallis's H test to compute if there were significant differences amongst the systems regarding pattern p_1. Kruskal-Wallis's H test outputted: p-value $= 0.000$, which is less than $0.05/4 = 0.0125$. This guarantees that there are significant differences amongst the systems. Then, we used Wilcoxon's Signed Rank test to compute pair ranks amongst the semantic data exchange systems, and we combined the results, which are shown in Table 10.3.

3.2. Sensitivity Analysis We first normalised the times of the previous step. Then, we used ReliefF to compute a ranking of parameters by setting, and the Majoritarian Compromise method to combine these rankings. The results are shown in Table 10.4.

4. Decision Making In this step, we used the previous results to make informed and statistically-sound decisions about the selected pattern and systems. Regarding the performance analysis, these decisions may be as follows:

- It is appealing to use m_9 since it spent the shortest CPU time in performing data exchange in the selected real-world pattern.
- It is not appealing to use m_{10} since it spent the longest CPU time in performing data exchange.

Regarding the sensitivity analysis, these decisions may be as follows:

- If we expect data exchange problems similar to p_1 and the number of individuals (I) is going to scale, all systems are much affected by the scaling of I.

10.5 Example with synthetic patterns

In this example, we used the same tool, virtual computer and software as in Section 10.4.

1. Initial Setup For this example, we selected the same ten semantic data exchange systems to test as in Section 10.4. Furthermore, we selected two synthetic patterns to test them: p_2 = Sink Properties; p_3 = Simplify Specialisation. We also selected the following values: $P_L = P_C = P_{IT} = P_{ID} = \{1, 2, 3, 4, 5\}$; $P_D = \{25, 50, 75, 100, 125\}$; $P_I = \{1, 250, 500, 750, 1000\}$; $P_{IO} = \{0\}$, which amounts to $15,625$ scenarios for each pattern. Finally, we selected CPU time as the performance variable.

2. Scenario Execution We selected the following thresholds: $\mu = 0.25$, $\delta = 5$, and we used our iterative method to execute the scenarios for each system and pattern. Table 10.5 shows our results. Note that we discarded a system if the ratio were greater than 25% ($\mu = 0.25$), or the iterations of the method were greater than five ($\delta = 5$). Therefore, we discarded the following systems in both patterns: m_1, m_2, m_5, m_8, and m_{10}. Note also that, in Table 10.5, the ratio between executed scenarios and parameters for the remaining systems does not fall below ten, i.e., all of them are greater than 70.

3.1. Performance Analysis In this step, we used Kruskal-Wallis's H test to compute if there were significant differences amongst the systems regarding patterns p_2, p_3 and $\{p_2, p_3\}$. Kruskal-Wallis's H test outputted the following p-values: for pattern p_2: p-value $= 2.381\,10^{-275}$; for pattern p_3: p-value $= 3.926\,10^{-142}$; for patterns $\{p_2, p_3\}$: p-value $= 0.000$. All of these p-values are less than $0.05/7 = 0.007$, which guarantees that there are significant differences amongst the systems. Then, we used Wilcoxon's Signed Rank test to compute pair ranks amongst the systems, and then we combined the results. The final results are shown in Table 10.6.

Pattern	p_2				p_3			
System	Size	Ratio	Iter.	Accept	Size	Ratio	Iter.	Accept
m_1	7812	49,99%	1	×	7805	49,95%	1	×
m_2	7755	49,63%	1	×	7622	48,78%	1	×
m_3	492	3,15%	3	✓	128	0,82%	2	✓
m_4	775	5,08%	3	✓	286	1,83%	2	✓
m_5	7736	49,50%	1	×	7628	48,82%	1	×
m_6	1455	9,32%	2	✓	610	3,90%	3	✓
m_7	1739	11,14%	3	✓	128	0,82%	2	✓
m_8	7644	48,92%	1	×	7648	48,95%	1	×
m_9	3777	24,17%	4	✓	905	5,79%	3	✓
m_{10}	6374	40,79%	1	×	4670	29,88%	1	×

Table 10.5: *Results of the iterative method (synthetic patterns).*

Patterns	Rankings
{p_2}	$m_7 = m_9 > m_4 > m_6 > m_3$
{p_3}	$m_7 > m_6 > m_9 > m_4 > m_3$
{p_2, p_3}	$m_7 = m_9 > m_4 > m_6 > m_3$

Table 10.6: *Results of the performance analysis (synthetic patterns).*

3.2. Sensitivity Analysis We normalised the times of the previous step, and used ReliefF to compute a ranking of parameters by setting, and the Majoritarian Compromise method to combine these rankings by systems and patterns. The final results are shown in Table 10.7.

4. Decision Making We used the previous results to make informed and statistically-sound decisions about the selected patterns and systems. Regarding the performance analysis, these decisions may be as follows:

- It is appealing to use m_3 since it spent the shortest CPU time in performing data exchange in all selected patterns.
- It is not appealing to use m_7 since it spent the longest CPU time in performing data exchange.

Systems	Patterns	Rankings
{m_3}	{p_2}	L > C > I > I_O > I_D > I_T > D
{m_4}	{p_2}	I > L > C > I_O > I_T > D > I_D
{m_6}	{p_2}	L > C > I > I_O > I_T > I_D > D
{m_7}	{p_2}	I > D > L > I_T > C > I_D > I_O
{m_9}	{p_2}	L > C > I > I_O > I_D > D > I_T
{m_3}	{p_3}	I > I_D > I_O > D > L > I_T > C
{m_4}	{p_3}	I > D > I_O > L > I_D > C > I_T
{m_6}	{p_3}	I > I_D > I_O > C > L > D > I_T
{m_7}	{p_3}	I > I_D > I_O > D > L > C > I_T
{m_9}	{p_3}	I > L > C > I_O > D > I_D > I_T
{m_3}	{p_2, p_3}	I > L > I_O > I_D > C > D > I_T
{m_4}	{p_2, p_3}	I > L > I_O > D > C > I_T > I_D
{m_6}	{p_2, p_3}	I > L > C > I_O > I_D > I_T > D
{m_7}	{p_2, p_3}	I > D > L > I_D > I_O > I_T > C
{m_9}	{p_2, p_3}	L > I > C > I_O > I_D > D > I_T
{m_3, m_4, m_6, m_7, m_9}	{p_2}	L > I > C > I_O > I_T > D > I_D
{m_3, m_4, m_6, m_7, m_9}	{p_3}	I > I_O > I_D > L > D > C > I_T
{m_3, m_4, m_6, m_7, m_9}	{p_2, p_3}	I > I_O > I_D > L > D > C > I_T

Table 10.7: *Results of the sensitivity analysis (synthetic patterns).*

Regarding the sensitivity analysis, these decisions may be as follows:

- If we expect data exchange problems similar to p_2 and the number of individuals (I) is going to scale, it is better to use systems m_3, m_6 or m_9 since they are not much affected by the scaling of I.
- If we expect data exchange problems similar to p_3 and the number of individuals (I) is going to scale, the CPU time shall be affected since all systems are much affected by the scaling of I.

10.6 Summary

In this chapter, we have presented the validation of our proposal to automatically generate SPARQL executable mappings, which is based on a repository that comprises four real-world and 3,780 synthetic data exchange problems. Furthermore, we have described an evaluation methodology that helps

software engineers make informed and statistically-sound decisions regarding semantic data exchange systems.

Part IV

Final Remarks

Chapter 11

Conclusions

I have seen too much not to know that the impression of a woman may be more valuable than the conclusion of an analytical reasoner.

Sir Arthur Conan Doyle, The Man with the Twisted Lip (1891)

In this dissertation, we present MostoDE, a proposal to automatically generate SPARQL executable mappings in the context of ontologies that are represented in quite a complete subset of the OWL Lite profile. These mappings are executed over a source ontology using a SPARQL query engine, and the source data are exchanged into data of a target ontology.

Our proposal takes a data exchange problem as input, which comprises a source ontology, a target ontology, and a number of correspondences between them, and it outputs a SPARQL executable mapping for each correspondence of the data exchange problem. These SPARQL executable mappings are generated by means of kernels, each of which describes the structure of a subset of data in the source ontology that needs to be exchanged as a whole, and the structure of a subset of data in the target ontology that needs to be created as a whole: if more or less data are considered, then the exchange would be incoherent.

Correspondences are inherently ambiguous since there can be many different executable mappings that satisfy them, but generate different target data. We have validated our proposal using four real-world and seven synthetic

data exchange patterns, in which we check that the interpretation of correspondences that it encodes is coherent with expected results. This validation relies on a repository that has been constructed using MostoBM, a benchmark that is also described in this dissertation. This repository comprises 3,784 data exchange scenarios in which the time to execute our algorithms never exceeded one second, and precision and recall were 100% in all cases except one, in which our proposal does not exchange data correctly. These results suggest that our proposal seems promising enough for real-world data exchange problems, that it is very efficient in practice, and that the interpretation of correspondences that our executable mappings encode is appropriate.

In addition to data exchange, our executable mappings have more potential uses in other fields, and it is our plan to research on them as future work: in the context of data integration, our executable mappings can be used to reformulate a target query into a single query over source data models [45]; in the context of query answering, which consists of retrieving appropriate data as a response to a query that is posed over a target data model that is physically divided into different sources, our executable mappings can be used to answer target queries [57, 88]; in the context of web service composition, our executable mappings can be used to combine the data of a number of existing web services [113].

Part V

Appendices

Appendix A

Subset of the OWL 2 Lite profile

Our proposal is based on the OWL 2 Lite profile specification, which provides 43 constructs that are classified into the following groups: RDF Schema features, equality, restricted cardinality, property characteristics, general property restrictions, class intersection, and meta information. In this appendix, we analyse these groups and describe the ones with which our proposal cannot deal. The appendix is organised as follows: in Section A.1, we introduce it; Section A.2 presents the RDF features group; Section A.3 describes the equalities and inequalities; Section A.4 deals with the constructs regarding restrictions of cardinalities; in Section A.5, we describe characteristics of properties; Section A.6 presents general property restrictions; Section A.7 describes the constructs that deal with the intersection of classes; in Section A.8, we present the constructs that deal with defining meta information; finally, we summarise the appendix in Section A.9.

A.1 Introduction

The OWL 2 Lite profile specification provides a number of constructs that are classified into the following groups [14]: RDF Schema features, equality, restricted cardinality, property characteristics, general property restrictions, class intersection, and meta information. We analyse them in the following sections. The conclusion is that the only constructs with which our proposal cannot deal are zero-cardinality restrictions, general property restrictions, and intersection of restrictions. This amounts to three out of 43 constructs.

A.2 RDF Schema Features

This group includes the following constructs, namely: owl:Individual, owl:Class, rdfs:subClassOf, rdf:Property, rdfs:subPropertyOf, rdfs:range, and rdfs:domain. The unique construct that our proposal ignores is owl:Individual, since individuals are retrieved or constructed by means of the executable mappings we generate, but needs not be dealt with explicitly in our proposal.

A.3 Equality and Inequality

In this group, we distinguish two subgroups. On the one hand, the first subgroup comprises the following constructs, namely: owl:equivalentClass, and owl:equivalentProperty. These two constructs need not be dealt with explicitly, since they are actually abbreviations. If an ontology contains a triple of the form $(c_1, \text{owl:equivalentClass}, c_2)$, it can be replaced by the following triples, with which we can deal natively:

$(c_1, \text{rdfs:subClassOf}, c_2)$
$(c_2, \text{rdfs:subClassOf}, c_1)$

Similarly, if a triple of the form $(p_1, \text{owl:equivalentProperty}, p_2)$ is contained in an ontology, we may replace it by the following triples, with which we can also deal natively:

$(p_1, \text{rdfs:subPropertyOf}, p_2)$
$(p_2, \text{rdfs:subPropertyOf}, p_1)$

On the other hand, the second subgroup comprises the following constructs, namely: owl:sameAs, owl:differentFrom, owl:AllDifferent, and owl:distinctMembers. The constructs in this second subgroup deal with individuals; thus we may ignore them for data exchange purposes.

A.4 Restricted Cardinality

The constructs in this group allow to restrict the cardinality of a property, namely: owl:minCardinality, owl:maxCardinality, and owl:cardinality.

The owl:minCardinality construct can restrict the minimum cardinality of a property p with respect to a class c to be zero as follows:

(c, rdfs:subClassOf, _:x)
(_:x, rdf:type, owl:Restriction)
(_:x, owl:minCardinality, "0"^^xsd:int)
(_:x, owl:onProperty, p)

Note that, by default, all data and object properties have a minimum cardinality of zero. Therefore, it does not require any special treatment. Similarly, construct owl:minCardinality can restrict the minimum cardinality of a property p with respect to a class c to be one as follows:

(c, rdfs:subClassOf, _:x)
(_:x, rdf:type, owl:Restriction)
(_:x, owl:minCardinality, "1"^^xsd:int)
(_:x, owl:onProperty, p)

The previous set of triples can be replaced by (c, mosto:strongDomain, p), if property p has class c as domain, or by (c, mosto:strongRange, p), if property p has class c as range.

Construct owl:maxCardinality can be used to restrict the maximum cardinality of a property to zero or one. In the former case, the property is annulated, which is a case with which our proposal cannot deal; we, however, have not found this a practical limitation since it is not common at all to annulate a property. The later case is the default in the OWL 2 Lite profile; thus, it does not require any special treatment.

Construct owl:cardinality is a shorthand to combine the previous constructs; thus, neither does it require special treatment.

A.5 Property Characteristics

Constructs owl:ObjectProperty and owl:DatatypeProperty are included in this group, with which we deal natively; owl:TransitiveProperty must actually be dealt with by a semantic-web reasoner, i.e., we can assume that

their semantics have been made explicit before using our proposal; the rest of constructs are shorthands, namely: owl:inverseOf, owl:SymmetricProperty, owl:FunctionalProperty, and owl:InverseFunctionalProperty.

If an ontology contains a subset of triples of the form:

$(p_1, \text{rdfs:domain}, c_1)$
$(p_1, \text{rdfs:range}, c_2)$
$(p_1, \text{owl:inverseOf}, p_2)$

we can transform them into the following triples, with which we can deal natively:

$(p_2, \text{rdfs:domain}, c_2)$
$(p_2, \text{rdfs:range}, c_1)$

Furthermore, we can transform $(p, \text{rdf:type}, \text{owl:SymmetricProperty})$ into a triple of the form $(p, \text{owl:inverseOf}, p)$. Similarly, we can transform a triple of the form $(p, \text{rdf:type}, \text{owl:FunctionalProperty})$ into the following triples before using our proposal:

(_:x, rdf:type, owl:Restriction)
(_:x, owl:minCardinality, "0"^^xsd:int)
(_:x, owl:maxCardinality, "1"^^xsd:int)
(_:x, onProperty, p)

Finally, if we find a subset of triples of the form:

$(p_1, \text{owl:inverseOf}, p_2)$
$(p_2, \text{rdf:type}, \text{owl:InverseFunctionalProperty})$

we can transform it into the following triples before applying our proposal:

$(p_1, \text{rdf:type}, \text{owl:FunctionalProperty})$
$(p_2, \text{rdf:type}, \text{owl:FunctionalProperty})$

A.6 General Property Restrictions

This group includes a number of constructors that allow to express general constraints on properties, namely: owl:Restriction, owl:onProperty, owl:allValuesFrom, and owl:someValuesFrom. We cannot deal with these constructs in a general problem, but only in the cases that we have mentioned in the previous sections, i.e., functional properties and cardinality restrictions.

A.7 Class Intersection

This group includes a single construct: owl:intersectionOf. Since it deals with the individuals of a class, our proposal does not need to pay attention to the triples that include this construct.

A.8 Meta Information

This category comprises three groups of constructs in the OWL 2 Lite profile, namely: header information, versioning, and annotation types. They include constructs like owl:Ontology, owl:imports, owl:versionInfo, or owl:backwardCompatibleWith, to mention a few. They provide meta information about an ontology, which make them irrelevant for data exchange purposes.

A.9 Summary

In this appendix, we have analysed the 43 constructs provided by the OWL 2 Lite profile specification. The main conclusion is that our proposal cannot deal with zero-cardinality restrictions, general property restrictions, and intersection of restrictions.

Bibliography

[1] B. Aleman-Meza, F. Hakimpour, I. B. Arpinar, and A. P. Sheth. *SwetoDblp ontology of computer science publications*. *J. Web Sem.*, 5(3): 151–155, 2007.

[2] B. Alexe, L. Chiticariu, R. J. Miller, D. Pepper, and W. C. Tan. *Muse: a system for understanding and designing mappings*. In *SIGMOD*, pages 1281–1284, 2008.

[3] B. Alexe, L. Chiticariu, R. J. Miller, and W. C. Tan. *Muse: Mapping understanding and design by example*. In *ICDE*, pages 10–19, 2008.

[4] B. Alexe, W. C. Tan, and Y. Velegrakis. *STBenchmark: Towards a benchmark for mapping systems*. *PVLDB*, 1(1):230–244, 2008.

[5] F. Alkhateeb, J.-F. Baget, and J. Euzenat. *Extending SPARQL with regular expression patterns (for querying RDF)*. *J. Web Sem.*, 7(2):57–73, 2009.

[6] S. Amano, L. Libkin, and F. Murlak. *XML schema mappings*. In *PODS*, pages 33–42, 2009.

[7] G. Antoniou and F. van Harmelen. *A Semantic Web primer*. The MIT Press, 2008

[8] M. Arenas, P. Barceló, and J. L. Reutter. *Query languages for data exchange: Beyond unions of conjunctive queries*. *Theory Comput. Syst.*, 49(2):489–564, 2011.

[9] M. Arenas and L. Libkin. *XML data exchange: Consistency and query answering*. *J. ACM*, 55(2):1–72, 2008.

[10] J. L. Arjona, R. Corchuelo, D. Ruiz, and M. Toro. *From wrapping to knowledge*. *IEEE Trans. Knowl. Data Eng.*, 19(2):310–323, 2007.

[11] *ARQ: A SPARQL processor*, January 2012.

[12] S. Auer, J. Lehmann, and S. Hellmann. *LinkedGeoData: Adding a spatial dimension to the Web of Data*. In *ISWC*, pages 731–746, 2009.

[13] M. Bada, R. Stevens, C. A. Goble, Y. Gil, M. Ashburner, J. M. Cherry, J. A. Blake, M. A. Harris, and S. Lewis. *A short study on the success of the Gene Ontology*. *J. Web Sem.*, 1(2):235–240, 2004.

[14] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. *OWL web ontology language reference*. Technical report, W3C, 2004.

[15] Z. Bellahsene, A. Bonifati, and E. Rahm, editors. *Schema matching and mapping*. Springer, 2011.

[16] T. Berners-Lee, J. Hendler, and O. Lassila. *The Semantic Web*. *Scientific American*, 284(5):34–43, 2001.

[17] P. A. Bernstein, T. J. Green, S. Melnik, and A. Nash. *Implementing mapping composition*. *VLDB J.*, 17(2):333–353, 2008.

[18] P. A. Bernstein and L. M. Haas. *Information integration in the enterprise*. *Commun. ACM*, 51(9):72–79, 2008.

[19] P. A. Bernstein and S. Melnik. *Model management 2.0: Manipulating richer mappings*. In *SIGMOD*, pages 1–12, 2007.

[20] C. Bizer. *The emerging web of Linked Data*. *IEEE Intelligent Systems*, 24(5):87–92, 2009.

[21] C. Bizer, P. Boncz, M. L. Brodie, and O. Erling. *The meaningful use of Big Data: Four perspectives - four challenges*. *SIGMOD Record*, 40(4): 56–60, 2011.

[22] C. Bizer, T. Heath, and T. Berners-Lee. *Linked Data: The story so far*. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[23] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. *DBpedia: A crystallization point for the Web of Data*. *J. Web Sem.*, 7(3):154–165, 2009.

[24] C. Bizer and A. Schultz. *The Berlin SPARQL benchmark*. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.

[25] C. Bizer and A. Schultz. *The R2R framework: Publishing and discovering mappings on the web*. In *COLD*, 2010

[26] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. *Contextualizing ontologies*. *J. Web Sem.*, 1(4):325–343, 2004.

[27] D. Brickley and R. Guha. *RDF vocabulary description language 1.0: RDF schema*. Technical report, W3C, 2004.

[28] D. W. Cearley, W. Andrews, and N. Gall. *Finding and exploiting value in semantic technologies on the Web*. Technical report, Gartner, 2007

[29] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. *Web services description language (WSDL) version 2.0 part 1: Core language*. Technical report, W3C, 2007.

[30] N. Choi, I.-Y. Song, and H. Han. *A survey on ontology mapping*. *SIGMOD Record*, 35(3):34–41, 2006.

[31] W. G. Cochran. *Sampling techniques*. John Wiley & Sons, 1977

[32] I. F. de Viana, I. Hernández, P. Jiménez, C. R. Rivero, and H. A. Sleiman. *Integrating deep-web information sources*. In *PAAMS*, pages 311–320, 2010.

[33] A. Deutsch, A. Nash, and J. B. Remmel. *The chase revisited*. In *PODS*, pages 149–158, 2008.

[34] D. Dou, D. V. McDermott, and P. Qi. *Ontology translation on the Semantic Web*. *J. Data Semantics*, 2:35–57, 2005.

[35] J. Euzenat. *An API for ontology alignment*. In *ISWC*, pages 698–712, 2004.

[36] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag, 2007

[37] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. *Data exchange: Semantics and query answering*. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[38] G. Flouris, D. Manakanatas, H. Kondylakis, D. Plexousakis, and G. Antoniou. *Ontology change: Classification and survey*. *Knowledge Eng. Review*, 23(2):117–152, 2008.

[39] M. J. Franklin, A. Y. Halevy, and D. Maier. *From databases to dataspaces: a new abstraction for information management*. *SIGMOD Record*, 34(4): 27–33, 2005.

[40] A. Fuxman, M. A. Hernández, C. T. H. Ho, R. J. Miller, P. Papotti, and L. Popa. *Nested mappings: Schema mapping reloaded*. In *VLDB*, pages 67–78, 2006.

[41] J. M. García, C. R. Rivero, D. Ruiz, and A. R. Cortés. *On using semantic web query languages for semantic web services provisioning*. In *SWWS*, pages 67–71, 2009

[42] Y. Guo, Z. Pan, and J. Heflin. *LUBM: A benchmark for OWL knowledge base systems*. *J. Web Sem.*, 3(2-3):158–182, 2005.

[43] Y. Guo, A. Qasem, Z. Pan, and J. Heflin. *A requirements driven framework for benchmarking semantic web knowledge base systems*. *IEEE Trans. Knowl. Data Eng.*, 19(2):297–309, 2007.

[44] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. *Clio grows up: from research prototype to industrial tool*. In *SIGMOD*, pages 805–810, 2005.

[45] A. Y. Halevy. *Answering queries using views: A survey*. *VLDB J.*, 10(4): 270–294, 2001.

[46] A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. *The Piazza peer data management system*. *IEEE Trans. Knowl. Data Eng.*, 16(7):787–798, 2004.

[47] S. Harris and A. Seaborne. *SPARQL 1.1 query language*. Technical report, W3C, 2012.

[48] H. He, W. Meng, C. T. Yu, and Z. Wu. *Automatic integration of web search interfaces with WISE-Integrator*. *VLDB J.*, 13(3):256–273, 2004.

[49] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a global data space*. Morgan & Claypool, 2011

[50] M. Hollander and D. A. Wolfe. *Non-parametric statistical methods*. Wiley-Interscience, 1999

[51] J. E. Hopcroft and R. E. Tarjan. *Efficient algorithms for graph manipulation [H] (Algorithm 447)*. *Commun. ACM*, 16(6):372–378, 1973.

[52] *Z formal specification notation: syntax, type system and semantics*, 2002

[53] *The Jena framework*, January 2012.

[54] M. Klusch, B. Fries, and K. P. Sycara. *OWLS-MX: A hybrid semantic web service matchmaker for OWL-S services*. *J. Web Sem.*, 7(2):121–133, 2009.

[55] G. Klyne and J. J. Carroll. *Resource description framework (RDF): Concepts and abstract syntax*. Technical report, W3C, 2004.

[56] G. Kobilarov, T. Scott, Y. Raimond, S. Oliver, C. Sizemore, M. Smethurst, C. Bizer, and R. Lee. *Media meets Semantic Web: How the BBC uses DBpedia and Linked Data to make connections*. In *ESWC*, pages 723–737, 2009.

[57] A. Langegger, W. Wöß, and M. Blöchl. *A semantic web middleware for virtual data integration on the web*. In *ESWC*, pages 493–507, 2008.

[58] M. Lenzerini. *Data integration: A theoretical perspective*. In *PODS*, pages 233–246, 2002.

[59] *Linked Open Data cloud*, January 2012.

[60] A. Maedche, B. Motik, N. Silva, and R. Volz. *MAFRA: A MApping FRAmework for distributed ontologies*. In *EKAW*, pages 235–250, 2002.

[61] B. Marnette, G. Mecca, and P. Papotti. *Scalable data exchange with functional dependencies*. *PVLDB*, 3(1):105–116, 2010.

[62] B. Marnette, G. Mecca, P. Papotti, S. Raunich, and D. Santoro. *++Spicy: an opensource tool for second-generation schema mapping and data exchange*. *PVLDB*, 4(12):1438–1441, 2011.

[63] G. Mecca, P. Papotti, and S. Raunich. *Core schema mappings*. In *SIGMOD*, pages 655–668, 2009.

[64] G. Mecca, P. Papotti, S. Raunich, and M. Buoncristiano. *Concise and expressive mappings with +Spicy*. *PVLDB*, 2(2):1582–1585, 2009.

[65] S. L. S. Mergen and C. A. Heuser. *Data translation between taxonomies*. In *CAiSE*, pages 111–124, 2006.

[66] A. Miles and S. Bechhofer. *SKOS: Simple knowledge organization system*. Technical report, W3C, 2009.

[67] D. E. Miller and J. T. Kunce. *Prediction and statistical overkill revisited*. *Measurement and Evaluation in Guidance*, 6(3):157–163, 1973

[68] R. G. Miller. *Simultaneous statistical inference*. Springer, 1981

[69] A. Mocan and E. Cimpian. *An ontology-based data mediation framework for semantic environments*. *Int. J. Semantic Web Inf. Syst.*, 3(2): 69–98, 2007.

[70] *MostoBM: benchmarking semantic data exchange systems*, January 2012.

[71] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. *OWL 2 web ontology language profiles*. Technical report, W3C, 2009.

[72] B. Motik, I. Horrocks, and U. Sattler. *Bridging the gap between OWL and relational databases*. *J. Web Sem.*, 7(2):74–89, 2009.

[73] M. Mrissa, C. Ghedira, D. Benslimane, Z. Maamar, F. Rosenberg, and S. Dustdar. *A context-based mediation approach to compose semantic web services*. *ACM Trans. Internet Techn.*, 8(1), 2007.

[74] N. F. Noy. *Semantic integration: A survey of ontology-based approaches*. *SIGMOD Record*, 33(4):65–70, 2004.

[75] N. F. Noy and M. C. A. Klein. *Ontology evolution: Not the same as schema evolution*. *Knowl. Inf. Syst.*, 6(4):428–440, 2004.

[76] B. Omelayenko. *Integrating vocabularies: Discovering and representing vocabulary maps*. In *ISWC*, pages 206–220, 2002.

[77] *Oracle database semantic technologies*, January 2012.

[78] C. R. Osuna, D. Ruiz, R. Corchuelo, and J. L. Arjona. *SPARQL query splitter: query translation between different contexts*. In *JISBD*, pages 320–323, 2009

[79] *OWLIM semantic repositories*, January 2012.

[80] F. S. Parreiras, S. Staab, S. Schenk, and A. Winter. *Model driven specification of ontology translations*. In *ER*, pages 484–497, 2008.

[81] C. Pedrinaci and J. Domingue. *Toward the next wave of services: Linked services for the Web of Data*. *J. UCS*, 16(13):1694–1719, 2010.

[82] *Pellet: OWL 2 reasoner for Java*, January 2012.

[83] M. Petropoulos, A. Deutsch, Y. Papakonstantinou, and Y. Katsis. *Exporting and interactively querying web service-accessed sources: The CLIDE system*. *ACM Trans. Database Syst.*, 32(4), 2007.

[84] A. Polleres and D. Huynh. *Special issue: The Web of Data*. *J. Web Sem.*, 7(3):135, 2009.

[85] A. Polleres, F. Scharffe, and R. Schindlauer. *SPARQL++ for mapping between RDF vocabularies*. In *ODBASE*, pages 878–896, 2007.

[86] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. *Translating web data*. In *VLDB*, pages 598–609, 2002.

[87] H. Qin, D. Dou, and P. LePendu. *Discovering executable semantic mappings between ontologies*. In *ODBASE*, pages 832–849, 2007.

[88] B. Quilitz and U. Leser. *Querying distributed RDF data sources with SPARQL*. In *ESWC*, pages 524–538, 2008.

[89] A. Raffio, D. Braga, S. Ceri, P. Papotti, and M. A. Hernández. *Clip: A visual language for explicit schema mappings*. In *ICDE*, pages 30–39, 2008.

[90] E. Rahm and P. A. Bernstein. *A survey of approaches to automatic schema matching*. *VLDB J.*, 10(4):334–350, 2001.

[91] J. Ressler, M. Dean, E. Benson, E. Dorner, and C. Morris. *Application of ontology translation*. In *ISWC*, pages 830–842, 2007.

[92] C. R. Rivero. *From queries to search forms: an implementation*. *IJCAT*, 33(4):264–270, 2008.

[93] C. R. Rivero, R. Z. Frantz, D. Ruiz, and R. Corchuelo. *On using high-level structured queries for integrating deep-web information sources*. In *SERP*, pages 330–341, 2011

[94] C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo. *Generating SPARQL executable mappings to integrate ontologies*. In *ER*, pages 118–131, 2011.

[95] C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo. *Mosto: Generating SPARQL executable mappings between ontologies*. In *ER Workshops*, pages 345–348, 2011.

[96] C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo. *On benchmarking data translation systems for semantic-web ontologies*. In *CIKM*, pages 1613–1618, 2011.

[97] C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo. *On using database techniques for generating ontology mappings*. In *SWWS*, pages 136–141, 2011

[98] C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo. *A reference architecture for building semantic-web mediators*. In *CAiSE Workshops*, pages 330–341, 2011.

[99] M. Robnik-Sikonja and I. Kononenko. *Theoretical and empirical analysis of ReliefF and RReliefF*. *Machine Learning*, 53(1-2):23–69, 2003.

[100] A. D. Sarma, X. Dong, and A. Y. Halevy. *Bootstrapping pay-as-you-go data integration systems*. In *SIGMOD*, pages 861–874, 2008.

[101] F. Scharffe and D. Fensel. *Correspondence patterns for ontology alignment*. In *EKAW*, pages 83–92, 2008.

[102] F. Scharffe, J. Euzenat, and D. Fensel. *Towards design patterns for ontology alignment*. In *ACM SAC*, pages 2321–2325, 2008.

[103] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. *SP^{2}Bench: A SPARQL performance benchmark*. In *ICDE*, pages 222–233, 2009.

[104] L. Serafini and A. Tamilin. *Instance migration in heterogeneous ontology environments*. In *ISWC*, pages 452–465, 2007.

[105] M. R. Sertel and B. Yilmaz. *The majoritarian compromise is majoritarian-optimal and subgame-perfect implementable*. *Social Choice and Welfare*, 16:615–627, 1999.

[106] N. Shadbolt, T. Berners-Lee, and W. Hall. *The Semantic Web revisited*. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

[107] P. Shvaiko and J. Euzenat. *Ontology matching: State of the art and future challenges*. *IEEE Trans. Knowl. Data Eng.*, In press, 2012.

[108] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. *Pellet: A practical OWL-DL reasoner*. *J. Web Sem.*, 5(2):51–53, 2007.

[109] L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic. *User-driven ontology evolution management*. In *EKAW*, pages 285–300, 2002.

[110] *TDB RDF store*, January 2012.

[111] B. ten Cate, L. Chiticariu, P. G. Kolaitis, and W. C. Tan. *Laconic schema mappings: Computing the core with SQL queries*. *PVLDB*, 2(1):1006–1017, 2009.

[112] H. J. ter Horst. *Completeness, decidability and complexity of entailment for RDF schema and a semantic extension involving the OWL vocabulary*. *J. Web Sem.*, 3(2-3):79–115, 2005.

[113] S. Thakkar, J. L. Ambite, and C. A. Knoblock. *Composing, optimizing, and executing plans for bioinformatics web services*. *VLDB J.*, 14(3): 330–353, 2005.

[114] Y. Theoharis, Y. Tzitzikas, D. Kotzinos, and V. Christophides. *On graph features of semantic web schemas*. *IEEE Trans. Knowl. Data Eng.*, 20(5): 692–702, 2008.

[115] *Semantic web development tools*, January 2012.

[116] D. Wood. *Linking enterprise data*. Springer, 2010

[117] Z. Wu, G. Eadon, S. Das, E. I. Chong, V. Kolovski, J. Srinivasan, and M. Annamalai. *Implementing an inference engine for RDFS/OWL constructs and user-defined rules in Oracle*. In *ICDE*, pages 1239–1248, 2008.

This document was typeset on March 24, 2012 at 17:33 using class RC-BOOK α2.13 for LaTeX2ε. As of the time of writing this document, this class is not publicly available since it is in alpha version. Only members of The Distributed Group are using it to typeset their documents. Should you be interested in giving forthcoming public versions a try, please, do contact us at contact@tdg-seville.info