Pontifícia Universidade Católica do Rio de Janeiro

Eveline Russo Sacramento

An Approach for Dealing with Inconsistencies in Data Mashups

TESE DE DOUTORADO

Thesis presented to the Programa de Pós-Graduação em Informática of the Departamento de Informática, PUC-Rio as partial fulfillment of the requirements for the degree of Doutor em Ciências - Informática

Advisor: Prof. Marco Antonio Casanova

Rio de Janeiro
September 2015

PUC-Rio - Certificação Digital Nº 1112681/CA

Eveline Russo Sacramento

An Approach for Dealing with Inconsistencies in Data Mashups

Thesis presented to the Programa de Pós-Graduação em Informática of the Departamento de Informática, PUC-Rio as partial fulfillment of the requirements for the degree of Doutor.

Prof. Marco Antonio Casanova
Advisor
Departamento de Informática – PUC-Rio

Prof. Edward Hermann Hæusler
Departamento de Informática – PUC-Rio

Prof. José Antonio Fernandes de Macêdo
UFC

Prof. Geraldo Bonorino Xexéo
UFRJ

Prof. Luiz André Portes Paes Leme
UFF

Prof. José Eugênio Leal
Coordinator of the Centro Técnico Científico – PUC-Rio

Rio de Janeiro, September 11th, 2015

PUC-Rio - Certificação Digital Nº 1112681/CA

All rights reserved.

Eveline Russo Sacramento

Graduated in Computer Science at the Federal University of Ceará in 1990 and obtained her M.Sc. Degree in Computer Science from the Federal University of Minas Gerais in 1994.

Bibliographic data

Sacramento, Eveline Russo

An Approach for Dealing with Inconsistencies in Data Mashups / Eveline Russo Sacramento ; advisor: Marco Antonio Casanova. – 2015.

100 f. : il. ; 30 cm

Tese (Doutorado em Informática) – Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2015.

Inclui bibliografia

1. Informática – Teses. 2. Combinação de dados. 3. Verificação de restrições. 4. Lógica de Defaults. 5. Inconsistência. 6. Verificação de modelos. I. Casanova, Marco Antonio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CCD: 004

PUC-Rio - Certificação Digital Nº 1112681/CA

PUC-Rio - Certificação Digital Nº 1112681/CA

“O Captain! my Captain! our fearful trip is done, The ship has weather’d every rack, the prize we sought is won…” (Walt Whitman).

Acknowledgements

First of all to God, whom I used to talk and listen to very frequently when I was a child and whom I unfortunately forgot and ignored so many times when I became an adult. In the recent years, with all circumstances, I am learning how to talk to Him in another way, through my diligent mother in Heaven, and my sweet protector Maria Auxiliadora.

To my son Lucas, who is now 15 years old, but was a little boy when I began this almost never-ending journey. In those days, he used to be smaller than me in height, but *never* in wisdom or emotional equilibrium. He used to sit on my lap when I was studying, and remained there until he got all kisses and hugs that he deserved. He gave me so many positive advices, as our roles were reversed, which makes me think that I should had listened to him more often!

To my dear husband, love and friend Laércio, who finished his Phd in 2009 and has been patiently waiting since then, in order to take me with him to his post doctorate course. Thanks for all the emotional and financial support. I also thank you for all fights and scolding, for not letting me cry, and specially for not letting me have a sense of self-pity when things did not happen as I wanted.

To my father, my paragon of virtue, persistence and honesty. To my mother, for her love and care.

To my advisor Prof. Marco Antonio Casanova, not only for his support and great technical knowledge, but also for his comprehension and patience since the beginning. My dear professor, you are a reserved person that can be hard and sweet, precise and sincere at the same time. Thank you very much!

To my colleagues of FUNCEME – Fundação Cearense de Meteorologia e Recursos Hídricos, especially to the President Eduardo Sávio Passos Rodrigues Martins, for his support and serenity during the whole time.

To CAPES and PUC-Rio, for the grants which made this work possible.

PUC-Rio - Certificação Digital Nº 1112681/CA

To the secretary of Pós-Graduação em Informática of the Departamento de Informática, PUC-Rio, Regina Zanon, who helped me whenever I needed her attention.

To my special friend: Fernanda Lígia Rodrigues Lopes, who is like a sister to me in so many ocasions. To my best friends from UFC: Fernando Lemos, Bernadette Lóscio, Danusa Ribeiro, Fernando Wagner and Hélio Rodrigues, who crossed my path and are the best memories that I bring from the previous years. Thank you for your friendship, help, and also for believing that I deserved to be where I am now since the beginning.

To my friends from outside the University who distracted me and saved me from myself so many times!

Finally, this thesis is dedicated to those who really care about me.

PUC-Rio - Certificação Digital Nº 1112681/CA

Abstract

Sacramento, Eveline Russo; Casanova, Marco Antonio (Advisor). **An Approach for Dealing with Inconsistencies in Data Mashups.** Rio de Janeiro, 2015. 100p. D.Sc. Thesis - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

With the amount of data available on the Web, consumers can “mashup” and quickly integrate data from different sources belonging to the same application domain. However, data mashups constructed from independent and heterogeneous data sources may contain inconsistencies and, therefore, puzzle the user when observing the data. This thesis addresses the problem of creating a consistent data mashup from mutually inconsistent data sources. Specifically, it deals with the problem of testing, when data to be combined is inconsistent with respect to a predefined set of constraints. The main contributions of this thesis are: (1) the formalization of the notion of consistent data mashups by treating the data returned from the data sources as a default theory and considering a consistent data mashup as an extension of this theory; (2) a model checker for a family of Description Logics, which analyzes and separates consistent from inconsistent data and also tests the consistency and completeness of the obtained data mashups; (3) a heuristic procedure for computing such consistent data mashups.

Keywords

Data Mashup; Constraint Verification; Default Logic; Inconsistency; Model Checking.

PUC-Rio - Certificação Digital Nº 1112681/CA

Resumo

Sacramento, Eveline Russo; Casanova, Marco Antonio (Orientador). **Uma Abordagem para Lidar com Inconsistências em Combinações de Dados.** Rio de Janeiro, 2015. 100p. Tese de Doutorado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A grande quantidade de dados disponíveis na *Web* permite aos usuários combinarem e rapidamente integrarem dados provenientes de fontes diferentes, pertencentes ao mesmo domínio de aplicação. Entretanto, combinações de dados construídas a partir de fontes de dados independentes e heterogêneas podem gerar inconsistências e, portanto, confundir o usuário que faz uso de tais dados. Esta tese aborda o problema de criação de uma combinação consistente de dados a partir de fontes de dados mutuamente inconsistentes. Especificamente, aborda o problema de testar quando os dados a serem combinados são inconsistentes em relação a um conjunto pré-definido de restrições. As principais contribuições desta tese são: (1) a formalização da noção de combinação consistente de dados, tratando os dados retornados pelas fontes como uma Teoria de *Defaults* e considerando uma combinação consistente de dados como uma extensão desta teoria; (2) um verificador de modelos para uma família de Lógicas de Descrição, usado para analisar e separar os dados consistentes e inconsistentes, além de testar a consistência e completude das combinações de dados obtidas; (3) um procedimento heurístico para computar tais combinações consistentes de dados.

PUC-Rio - Certificação Digital Nº 1112681/CA

Palavras-chave

Combinação de dados; Verificação de restrições; Lógica de Defaults; Inconsistência; Verificação de modelos.

Table of Contents

1 Introduction 14
1.1 Motivation and Main Objectives 14
1.2 Contributions 14
1.3 Thesis Organization 15

2 Background 17
2.1 Data Integration 17
2.2 Query Answering 18
2.2.1 Consistent Query Answering 19
2.2.2 Paraconsistent Query Answering 20
2.3 Mashup Applications 21
2.4 Model Checking 24
2.5 The OWL 2 Web Ontology Language 25
2.5.1 Overview 25
2.5.2 The OWL 2 EL Profile 26
2.5.3 The OWL 2 QL Profile 27
2.6 DL-Lite Core with Arbitrary Number Restrictions 29
2.7 The Description Logic SROIQ and the OWL 2 Direct Semantics 31
2.8 Summary 32

3 A Case Study of Data Mashup 33
3.1 Brief Introduction to Linked Data 33
3.2 Description of the Data Sources 34
3.3 Description of the Data Mashup 35
3.4 Registering the Linked Data Sources 37
3.5 Querying the Linked Data Sources and Populating the Linked Data Mashup 38
3.6 Summary 41

PUC-Rio - Certificação Digital Nº 1112681/CA

4 Mashups as Default Theories 42
4.1 Default Logic 42
4.2 Mashup Default Theories 43
4.3 Case Study using Mashup Defaults 44
4.4 Correctness of the Default Model for Mashups 46
4.5 Summary 48

5 Computing Consistent Data Mashups 50
5.1 A Brute Force Method to Compute Extensions 50
5.2 Process Trees 54
5.3 Mashup Default Trees 57
5.3.1 Process Trees for Mashup Default Theories 57
5.3.2 Traversing a Mashup Default Tree 61
5.3.3 A Procedure to Compute Mashup Default Trees 64
5.3.4 Considerations on Heuristics for Building a Mashup Default Tree 73
5.4 Testing Consistency 74
5.4.1 Testing Consistency of OWL 2 Ontologies 74
5.4.2 Testing Consistency of Lightweight Ontologies 75
5.4.3 Testing Consistency of Lightweight Ontologies in the Context of Mashup Default Trees 82
5.5 Summary 84

6 Related Work 85
6.1 Consistent Mashups and Related Problems 85
6.2 Mashup Frameworks 88
6.3 Working with Defaults 90

7 Conclusions and Suggestions for Future Research 91
7.1 Conclusions 91
7.2 Suggestions for Future Research 93

PUC-Rio - Certificação Digital Nº 1112681/CA

PUC-Rio - Certificação Digital Nº 1112681/CA

List of Figures

Figure 1. The constraint graph $G(\Sigma_M)$ for $\Sigma_{M.}$ 36

Figure 2. A brute-force algorithm to determine all maximal sequences of a set of defaults. 52

Figure 3. A process tree for Example 4. 56

Figure 4. The typical topology of a mashup default tree with 4 defaults. 62

Figure 5. An implementation of the depth-first algorithm to determine all maximal subsets of a set of defaults in Δ. 65

Figure 6. The constraint graph $G(\Sigma_M)$ for $\Sigma_{M.}$ 66

Figure 7. The mashup default tree for Example 6. 67

Figure 8. The mashup default tree for Example 7. 69

Figure 9. The mashup default tree for Example 8. 71

Figure 10. A procedure to test the consistency of a set of positive assertions against a lightweight ontology. 77

PUC-Rio - Certificação Digital Nº 1112681/CA

List of Tables

Table 1. Common constraint types used in conceptual modeling. 30

Table 2. Set of mashup constraints Σ_M. 36

Table 3. Data sources and matching rules. 37

Table 4. Assertions expressing data from sources *S1*, *S2* and *S3*. 37

Table 5. Maximal consistent subsets of *S*. 37

Table 6. Maximal consistent subsets of assertions. 45

Table 7. Assertions in the extensions of the default theory *T*. 45

Table 8. A summary of the execution of Traverse_Mashup_Tree for Example 6. 68

Table 9. A summary of the execution of Traverse_Mashup_Tree for Example 7. 70

Table 10. A summary of the execution of Traverse_Mashup_Tree for Example 8. 72

Table 11. Combined Complexity of the Ontology Consistency Problem the OWL 2 Profiles. 75

PUC-Rio - Certificação Digital Nº 1112681/CA

1
Introduction

1.1
Motivation and Main Objectives

Applications that access data from several sources on the Web may face challenges with respect to dynamically accessing and combining data from such sources in a meaningful way (Hogan, 2011). Indeed, the broad use of the Web facilitates extracting and combining data from different sources, called *data mashups*, but it also increases the risks of creating and propagating "dirty" data, that is, data which is inconsistent, inaccurate, incomplete or stale (Fan et al., 2008). However, data mashups constructed from independent and heterogeneous data sources may contain inconsistencies and, therefore, puzzle the user when observing the data.

In this thesis, we investigate the problem of constructing consistent data mashups from data sources that are mutually inconsistent. We deal with a very specific problem of inconsistency: when data to be combined is inconsistent with respect to a predefined set of constraints. In fact, even if each data source returns data consistent with its own set of constraints, the combined data might be inconsistent.

Therefore, we address two questions. The first question is how to analyze data coming from different sources in order to identify and separate conflicting data. The second question is how to create a data mashup in such a way that the resulting data is consistent with a given set of constraints.

1.2
Contributions

This thesis contributes to the consistent consumption of integrated data generated from heterogeneous and independent data sources through data mashup applications. By separating consistent from inconsistent data, one can properly integrate data without generating conflicts.

PUC-Rio - Certificação Digital Nº 1112681/CA

The first contribution of this thesis is the formalization of the notion of consistent data mashups by treating the data returned from the data sources as a default theory and considering a consistent data mashup as an extension of this theory.

The second contribution is the definition of a model checker for a family of Description Logics, which analyzes and separates consistent from inconsistent data and tests the consistency and completeness of our approach.

The third contribution is a heuristic procedure to compute consistent data mashups, when the data sources return a positive set of assertions.

The formalization used in this thesis is based on a well-known family of Description Logics (Artale et al., 2009) and some concepts of Default Logic (Antoniou, 1999; Levesque et al., 2004). The heuristic procedure to compute consistent data mashups explores constraint graphs (Casanova et al., 2010) of the data mashup specification.

More specifically, we employ DL-Lite Core with arbitrary number restrictions, as the adopted dialect of Description Logics, as it is sufficient for our needs of expressivity. Furthermore, we restrict our attention to simple defaults, as they are sufficient for the purposes of formalizing our data mashups.

PUC-Rio - Certificação Digital Nº 1112681/CA

1.3 Thesis Organization

The remainder of this thesis is organized as follows.

Chapter 2 first presents a brief and an informal introduction to data integration, query answering and mashup applications, which are database problems directly related to the thesis. Then, it presents formal concepts used throughout the thesis: model checking, OWL2 and two dialects of Description Logics: $DL\text{-}Lite_{core}^{N}$ and $\mathcal{SROIQ}$.

Chapter 3 presents a case study in the area of Linked Data, which is our motivating example.

Chapter 4 formalizes the problem of building consistent data mashups in the context of Description Logics and also in the context of the theory of defaults. Firstly, it shows that the mapping of this problem between the two theories is correct and complete, as both approaches generate the same results. Finally, it

formalizes the case study of Chapter 3 as a Default Theory.

Chapter 5 first describes two existing methods that can be used for building consistent data mashups: a method based on brute force and an operational model, called *process trees*. Then, it presents a new method, called *mashup default trees*, which computes extensions of mashup default theories. This chapter also discusses the problem of testing the consistency of a set of assertions in the presence of a set of constraints for some variations of Description Logics, and presents an optimized procedure for DL-Lite Core with arbitrary number restrictions. Finally, it shows how to use a mashup default tree to construct a heuristic procedure for computing consistent data mashups.

Chapter 6 summarizes related work and Chapter 7 presents the main contributions of this thesis and suggests future research.

PUC-Rio - Certificação Digital Nº 1112681/CA

2 Background

In this chapter, we review concepts used in the remainder of this text. In the first part (Sections 2.1 to 2.3), we present a brief and informal introduction about data integration, query answering and mashup applications. In the second part (Sections 2.4 to 2.7), we briefly cover concepts about model checking, OWL2 and two dialects of Description Logics: DL-Lite Core with arbitrary number restrictions and $\mathcal{SROIQ}$.

2.1 Data Integration

There are an increasing number of available data sources, which are stored in different formats and range from highly structured, such as relational databases, to semi-structured, such as data on the Web. Many current applications need to access and combine information coming from such distinct data sources.

Data integration refers to the problem of combing data residing at autonomous and heterogeneous data sources and providing the user with a unified view of these data (Lenzerini, 2002). According to Lenzerini (2002), the main components of a data integration system are: (i) the *mediated view* or *global schema*, which provides both a conceptual representation of the application domain, and a reconciled, integrated, and virtual view of the underlying sources; (ii) the *source schema*, the schema of the sources where real data are stored; and (iii) the *mapping* that specifies the correspondences between the global schema and the source schemas.

It is clear that combining data from such sources creates several challenges that have to be faced. For example, an important issue in data integration is the possibility of dealing with inconsistencies among different data sources. Data sources may conflict with each other at three different levels (Wang et. al., 2011):

i. *Schema level*: sources are expressed using different data models or have different schemas within the same data model;

PUC-Rio - Certificação Digital Nº 1112681/CA

ii. *Data representation level*: data in the sources is represented in different natural languages or different measurement systems; and

iii. *Data value level*: there are discrepancies among the sources in data values that describe the same objects.

When two or more data sources conflict with each other at data value level, we have a problem called *data inconsistency*. This is the case, for example, when two objects, obtained from different data sources, are identified as versions of each other and some of the values of their corresponding properties differ (Wang et al., 2011). In this case, the solution is to choose, for each property of the object, a unique value from such different sets of values.

The treatment of inconsistencies arising from the integration of several data sources is a topic that has received increased attention lately and has become an important field of research in databases (De Amo et al., 2002; Lembo, 2004; Arenas et al., 1999; Wang et al., 2011). More recently, this problem is being considered in the Web context.

PUC-Rio - Certificação Digital Nº 1112681/CA

2.2 Query Answering

The ultimate goal of query answering is to provide trusted answers to a query, i.e., to compute the intersection of the answer sets obtained by evaluating the query over any database that satisfies the global schema (Lembo, 2004).

Query answering in the presence of inconsistent data requires deriving consistent information, despite the presence of data inconsistencies. This is the case when the data, stored at the sources, are not entirely incorrect. For example, an independent and autonomous data source may provide data that do not respect all established constraints. Since most of the data could satisfy such constraints, it seems unreasonable to consider the entire source as inconsistent.

Classical approaches for query answering do not handle such situations, i.e., they do not provide meaningful answers to the users in those cases. Roughly, two basic approaches are used for solving the inconsistency problem in knowledge bases (De Amo et al., 2002):

i. *Consistent-based approaches*: try to make an inconsistent theory consistent, either by revising it, or representing through a consistent semantics. So, its main goal is to avoid data contradictions.

ii. *Paraconsistent-based approaches*: the inconsistency is not rejected and inference methods can draw plausible conclusions from it. In this scenario, the inconsistencies are not removed, but the query answers can be marked as "consistent" or "inconsistent". Hence, such approach prevents information loss due to data cleaning, which may occur in the first approach (Arenas et al., 1999).

In addition to these two basic approaches, there are also hybrid approaches, which are based on formalisms that do not reject any information but, instead, associate degrees of belief, reliability or uncertainty to the data sources.

2.2.1 Consistent Query Answering

Consistent query answering means obtaining consistent information from possible inconsistent databases, in response to a user query. A consistent answer may be obtained by correcting data coming from distinct data sources, that is, by removing constraints violation with minimal change. In such approach, it is possible to deal with the problem of data inconsistency in three distinct ways:

i. *Conflict resolution*: not allowing inconsistencies and simply eliminating the duplicate data;
ii. *Belief revision*: trying to identify the data source with better belief and considering its information as the correct one (Wang et al., 2011); or
iii. *Voting*: voting to choose the most reliable data source and considering as correct the information it provides.

The process of conflict resolution can be complex, costly and nondeterministic. It happens because, sometimes, an application may not have enough time to resolve, in real time, all conflicts relevant to a query. Furthermore, removing data to restore consistency may lead to information loss, which is undesirable (Bertossi and Chomicki, 2003). For example, one may want to keep multiple addresses for a person, if it is not clear which is the correct one. The semantic problem that arises in this context is similar to the notion of database repair introduced by many works in the area of inconsistent databases (Arenas et al., 1999; Bertossi and Chomicki, 2003). A *repair* is another database that is consistent and minimally differs from the original database (Bertossi and Chomicki, 2003). However, a repair can be very expensive in terms of computing

PUC-Rio - Certificação Digital Nº 1112681/CA

power and complexity. Another problem is that we can potentially lose relevant data due to the repair process.

Belief revision is the process of changing beliefs to take into account a new piece of information. Its main goal is to identify the source with a better belief, for example, by looking at the qualifications of each data provider. An inconsistency may occur, for example, when older – and perhaps obsolete – data is compared with new, and perhaps up-to-date, data. Inconsistencies may also occur when data comes from a dubious provider, in which case both the old and the new data refer to the same situation, and the process consists in incorporating the new data into the set of old beliefs, while maintaining consistency with minimal change, i.e., the knowledge before and after the change should be as similar as possible. What makes belief revision a non-trivial operation is that it exists several different ways to revise data (or belief).

Finally, the process of voting consists in simply electing a data source based on a majority criterion. The elected data source is considered the most trustable by a group of users and, in the case of data inconsistency, it will be chosen to provide the correct data.

PUC-Rio - Certificação Digital Nº 1112681/CA

2.2.2 Paraconsistent Query Answering

Sometimes inconsistent information can be useful, unavoidable and even desirable. In these cases, it is important to find out which query answers, returned from the sources, are consistent with the constraints and which are not, given a set of predefined constraints. This problem is called *Paraconsistent Query Answering* (Villadsen, 2002) and it is inspired in Paraconsistent Logic, which is the subfield of Logic that is concerned with studying and developing “inconsistency-tolerant” systems of logic.

While in classical logic everything follows from an inconsistency, the meanings of some, or even all, of the logical operators in paraconsistent logics differ from classical logic, in order to block the explosion of consequences from inconsistencies (Villadsen, 2002). The intended semantics of classical logic is bivalent, i.e., it states that every declarative sentence expressing a proposition has exactly one truth-value, either true or false. So, a classical logic is called *two-valued* logic or *bivalent* logic.

In general, paraconsistent logics are *many-valued* and, as such, they differ from classical logic by the fundamental fact that they do not restrict the number of truth-values to only two. For example, in some situations, it can be quite important to retain all available information, as discarding inconsistent data could imply in losing information (De Amo et al., 2002). In this case, a paraconsistent approach would be recommended, as it consists in reasoning in the presence of inconsistencies, i.e., dealing with inconsistent data without fixing or discarding it.

2.3
Mashup Applications

In the Web 1.0, it was common to see portals regularly maintained by companies responsible to store and update their own data in order to share information with their users. However, such users could only get information through the provided products and services.

With the advent of Web 2.0, Web applications began publishing APIs (Application Programming Interfaces), which enabled software developers to easily integrate data and functions, instead of building them by themselves. However, the use of APIs brought a series of problems (Chen et al., 2009) that users had to face, such as the difficulties to find and combine the right data because of the lack of knowledge about APIs and relationships among them; and the need to find and read the specification of a new API before using it, so as to decipher its appropriate functions.

Mashup applications have an active role in the evolution of Web 2.0, as they allow users to access existing APIs to create new services for combining, visualizing, and aggregating data. The term *mashup* was borrowed from the area of pop music, where a mashup means a new song mixed from the vocal and instrumental tracks from two different source songs, which usually belong to different genres (Merril, 2006).

A *mashup*, in the area of Web development, is described as "*a Web page or Web application that uses and combines data, presentation or functionality from two or more sources to create new services*". So, a mashup implies easy and fast integration, using open APIs or data sources, to produce enriched results that were

PUC-Rio - Certificação Digital Nº 1112681/CA

not necessarily the original reason for producing the application or the raw source data.

Note that a manual mashup development still needs a user with programming skills and intimate knowledge about the schemes and semantics of the data sources, or the business protocol conventions for message exchanges. Thus, this task still requires advanced users (software developers). Nevertheless, a tool-assisted mashup development can simplify the integration of contents, application logic or user interfaces, as it is largely assisted by "intelligent" source components (Yu et al., 2008).

Mashup development tools are usually simple enough to be used by inexperienced end-users, as they generally do not require programming skills and can also speed up the overall mashup development process. Therefore, these tools contributed to a new vision of the Web, where even end-users were able to contribute. Examples of current mashup tools are: Yahoo Pipes and MapBuild. Some mashup tools like Google Mashup Editor (GME), Microsoft Popfly, and Intel Mash Maker were discontinued by their providers and are no longer available.

Mashups can be compared to others traditional integration approaches, such as application integration (Web Services) and data integration (Lenzerini, 2002). The main difference is that mashups are usually appropriated to very specific and short-living situations, and they are developed with the most modern Web technologies, such as AJAX/RESTful services and RSS/Atom feeds. Furthermore, mashups are usually appropriated for personal use, i.e., they are built for a small group of users (a small audience), and do not raise concerns about scalability, security or reliability. By contrast, traditional integration tools usually demand systematic and repeatable enterprise processes, which constitute business-critical applications with many users and a set of complex requirements that need to be followed (Yu et al., 2008).

There are three main types of mashups:

- *Business* or *enterprise mashups*, which define applications that combine heterogeneous data and applications from multiple sources for business purposes. They focus data into a single presentation and allow collaborative work among businesses and developers. These terms are often used to differentiate a business-related mashup from a data mashup,

PUC-Rio - Certificação Digital Nº 1112681/CA

which is usually used by the general public. Other differences are that the enterprise mashups include integration with the business computing environment, data governance, business intelligence (BI) / business analytics (BA) tools, use more sophisticated programming tools and also need more strict security policies.

- *Data mashups*, which integrate similar types of media and information from multiple heterogeneous sources into a single representation. The combination of such resources creates a new and distinct application that was not originally provided by either source, and which is suitable for general users.
- *Consumer mashups*, which combine data from multiple public sources in the browser and organize them through a simple browser user interface. Examples of consumer mashups are *WikipediaVision*, which shows in semi-real time where anonymous edits to Wikipedia are originating from (combining *Google Maps* for 2D visualization and *Wikipedia API* for seeing recent changes to Wikipedia); and *Google* Shopping, which allows users to search for products on online shopping websites and compare prices between different vendors.

Data mashups can be classified according to the type of data to be integrated:

- *Geographic or Cartographic data,* which provide geospatial data and applications for geovisualization, geolocation, etc. Such mashups may use APIs from Google (Google Maps), Microsoft (Virtual Earth), Yahoo (Yahoo Maps) and AOL (MapQuest).
- *Indexed data,* which provides documents, photos, videos, weblogs, etc. used by metasearch engines. These mashups may, for example, use APIs from Flickr and other social networks to provide images with associated metadata information (such as place, date, time, people in the photo, etc.).
- *Feeds, headlines, blogs and podcasts,* which provide Web content used by news aggregators. Such mashups may, for example, access data related to a topic from BBC, New York Times and Reuters (in RSS or Atom format) and create a personalized newspaper that meets a reader's particular interest.

Like any other data integration domain, mashup development raises

PUC-Rio - Certificação Digital Nº 1112681/CA

technical challenges that need to be addressed. Some of these challenges were already solved, while others are still open questions (Merril, 2006):

- *Semantic Meaning*: how to derive shared semantics from heterogeneous sources, before integrating legacy data sources.
- *Data Quality*: the need for data analysis and data cleaning before using tools for automated reasoning, as data can be subjected to inconsistency, incorrectness or intentionally misleading data entry.
- *Protection of data intellectual property*: the tradeoff between consumer privacy versus fair-use and free flow of information. Content providers might determine if their content is being used in an approved manner or not.

2.4 Model Checking

PUC-Rio - Certificação Digital Nº 1112681/CA

Model checking refers to the following problem: "Given a set of assertions ***A***, exhaustively and automatically check whether ***A*** is a model of a set of axioms that express a given specification". It provides an automated and formal technique to check the absence of errors, that is, to verify correctness properties. Model checking may also be considered an intelligent and effective debugging technique.

A *model checker*, the software tool that performs the model checking, examines all possible system scenarios in a systematic manner. It shows that a given system model satisfies or not a certain property (Baier and Katoen, 2008), i.e., it checks for property violation. It is algorithmic and often efficient, as the system has a finite number of states, despite the fact that reasoning can have an infinite behavior. In order to algorithmically solve a problem, both the model of the system and its specification need to be formulated in some precise mathematical language.

Several model-checking tools use LTL (Linear Temporal Logic), a temporal logic that is based on a linear-time perspective (or a slight variant of it), as the property specification language. LTL is called *linear* because the qualitative notion of time is *path-based*: at each moment of time there is only one possible successor state, and thus each time moment has a unique possible future. This

follows from the fact that the interpretation of LTL formulae is defined in terms of *sequences of states*.

Other model-checking tools use CTL (Computation Tree Logic), a logic that is based on a *branching-time* view, as at each moment of time there are many possible successor states, and thus each time moment may split into several possible futures. In this case, paths are obtained from a transition system that might be branching: a state may have several distinct direct successor states, and thus several computations may start in a state. So, CTL is a logic for formalizing *state-based* properties. The semantics of this kind of logic is not based on a linear notion of time – an infinite sequence of states - but on a branching notion of time - an infinite *tree of states* (Baier and Katoen, 2008). The difficulty is that the state graph can be immense. Abstraction can be helpful as it replaces the original state graph by a much smaller one, verifying the same formulae (Emerson, 2008).

Currently, there are many model checkers tools, such as SPIN (Holzmann, 2003), which is very popular and one of the most powerful tools for detecting software defects in concurrent system design. A historical background of the area of model checking is presented in Emerson (Emerson, 2008).

PUC-Rio - Certificação Digital Nº 1112681/CA

2.5 The OWL 2 Web Ontology Language

2.5.1 Overview

The OWL 2 Web Ontology Language (OWL 2, 2012a), informally known as OWL 2, is an extension and revision of the OWL Web Ontology Language developed by the W3C Web Ontology Working Group and published in 2004 (hereafter referred to as “OWL 1”). Like OWL 1, OWL 2 is designed to facilitate ontology development and sharing via Web, its ultimate goal is to make Web content more accessible to machines.

OWL 2 became a W3C Recommendation on Oct 27 2009. It is an ontology language for the Semantic Web with formally defined meaning. OWL 2 ontologies provide classes, properties, individuals, and data values and are stored as Semantic Web documents.

OWL 2 adds new functionality with respect to OWL 1. Some of the new features are syntactic sugar (e.g., disjoint union of classes), while others offer new expressivity, such as keys, property chains, richer datatypes, etc.

OWL 2 ontologies can be used along with information written in RDF; OWL 2 ontologies themselves are primarily exchanged as RDF documents. Any OWL 2 ontology can also be viewed as an RDF graph.

The specification of OWL 2 introduced several profiles or fragments for the language, by placing syntactic restrictions on the structure of OWL 2 ontologies (OWL 2, 2012b). These profiles are sub-languages of OWL 2 that offer important advantages in particular application scenarios. Each profile is a subset of the structural elements (syntactic subsets) that can be used in a conforming ontology, and it is also more restrictive than OWL DL. Finally, each profile trades off different aspects of OWL's expressive power, in return for different computational and/or benefits of implementation. In the next sections, we summarize two useful profiles that will be mentioned in this thesis.

2.5.2 The OWL 2 EL Profile

The OWL 2 EL profile is particularly useful in applications employing ontologies that contain very large numbers of properties or classes, and it captures the expressive power used by many such ontologies. The EL acronym reflects the profile's basis in the EL family of description logics (Baader et al., 2008) that provide only existential quantification.

OWL 2 EL places restrictions on the type of class restrictions that can be used in axioms. In particular, the following types of class restrictions are supported:

- Existential quantification to a class expression (ObjectSomeValuesFrom) or a data.
- Range (DataSomeValuesFrom).
- Existential quantification to an individual (ObjectHasValue) or a literal one (DataHasValue).
- Self-restriction (ObjectHasSelf).
- Enumerations involving a single individual (ObjectOneOf) or a single literal (DataOneOf).

PUC-Rio - Certificação Digital Nº 1112681/CA

- Intersection of classes (ObjectIntersectionOf) and data ranges (DataIntersectionOf).

OWL 2 EL supports the following axioms, all of which are restricted to the allowed set of class expressions:

- Class inclusion (SubClassOf).
- Class equivalence (EquivalentClasses).
- Class disjointness (DisjointClasses).
- Object property inclusion (SubObjectPropertyOf) with or without property chains, and data property inclusion (SubDataPropertyOf).
- Property equivalence (EquivalentObjectProperties and EquivalentDataProperties).
- Transitive object properties (TransitiveObjectProperty).
- Reflexive object properties (ReflexiveObjectProperty).
- Domain restrictions (ObjectPropertyDomain and DataPropertyDomain).
- Range restrictions (ObjectPropertyRange and DataPropertyRange).
- Assertions (SameIndividual, DifferentIndividuals, ClassAssertion, ObjectPropertyAssertion, DataPropertyAssertion, NegativeObjectPropertyAssertion and NegativeDataPropertyAssertion).
- Functional data properties (FunctionalDataProperty).
- Keys (HasKey).

OWL 2 EL enables polynomial time algorithms for all the standard reasoning tasks; it is particularly suitable for applications in which very large ontologies are needed, and which expressivity power can be traded for performance guarantees. So, the basic reasoning problems for OWL 2 EL – ontology consistency, class expression subsumption, and instance checking – can be performed in time that is polynomial with respect to the size of the ontology (Baader et al., 2008). Dedicated reasoning algorithms for this profile are available and have been demonstrated to be implementable in a highly scalable way.

2.5.3 The OWL 2 QL Profile

OWL 2 QL is based on the DL-Lite family of Description Logics (Calvanese et al., 2007). Several variants of DL-Lite have been described in the literature, and DL-Lite$_R$ provides the logical underpinning for OWL 2 QL.

PUC-Rio - Certificação Digital Nº 1112681/CA

DL-Lite$_R$ does not require the Unique Name Assumption (UNA), which is a simplifying assumption that says that different names (constants) always refer to different entities in the world, since making this assumption would have no impact on the semantic consequences of a DL-Lite$_R$ ontology.

More expressive variants of DL-Lite, such as DL-Lite$_A$, extend DL-Lite$_R$ with functional properties and can also be extended with keys. However, for query answering to remain in LOGSPACE, these extensions require UNA and need to impose certain global restrictions on the interaction between properties used in different types of axiom.

Basing OWL 2 QL on DL-Lite$_R$ avoids practical problems involved in the explicit axiomatization of UNA. Other variants of DL-Lite can also be supported on top of OWL 2 QL, but may require additional restrictions on the structure of ontologies.

OWL 2 QL supports the following axioms:

- Subclass axioms (SubClassOf).
- Class expression equivalence (EquivalentClasses).
- Class expression disjointness (DisjointClasses).
- Inverse object properties (InverseObjectProperties).
- Property inclusion (SubObjectPropertyOf not involving property chains and SubDataPropertyOf).
- Property equivalence (EquivalentObjectProperties and EquivalentDataProperties).
- Property domain (ObjectPropertyDomain and DataPropertyDomain).
- Property range (ObjectPropertyRange and DataPropertyRange).
- Disjoint properties (DisjointObjectProperties and DisjointDataProperties).
- Symmetric properties (SymmetricObjectProperty).
- Reflexive properties (ReflexiveObjectProperty).
- Irreflexive properties (IrreflexiveObjectProperty).
- Asymmetric properties (AsymmetricObjectProperty).
- Assertions other than individual equality assertions and negative property assertions (DifferentIndividuals, ClassAssertion, ObjectPropertyAssertion, and DataPropertyAssertion).

PUC-Rio - Certificação Digital Nº 1112681/CA

The OWL 2 QL profile is designed so that sound and complete query answering is in LOGSPACE (more precisely, in AC^0) with respect to the size of the data (assertions), using standard relational database technology. It is particularly suitable for applications in which relatively lightweight ontologies are used to organize large numbers of individuals and which is useful or necessary to access the data directly via relational queries (e.g., SQL).

2.6 DL-Lite Core with Arbitrary Number Restrictions

In this thesis, we adopt *DL-Lite core with arbitrary number restrictions* (Artale et al., 2009), denoted $DL\text{-}Lite^{N}_{core}$, a DL dialect that is useful for conceptual modeling. We refer to such ontologies as *lightweight ontologies* in the remainder of this text.

A *language L* in this dialect is characterized by a vocabulary *V*, consisting of a set of *object names*, a set of *atomic concepts*, a set of *atomic roles*, and the *bottom concept* ⊥.

The sets of *basic concept descriptions*, *concept descriptions,* and *role descriptions* of ***L*** are defined as follows:

- If *P* is an atomic role, then *P* and P^{-} (*inverse role*) are role descriptions.
- If *u* is an atomic concept or the bottom concept, and *p* is a role description, then *u* and (≥*n p*) (*at-least restriction*, where *n* is a positive integer) are basic concept descriptions and also concept descriptions.
- If *u* is a basic concept description, then ¬*u* (*negated concept*) is a concept description.

Furthermore, we abbreviate "¬(≥ *n+1 p*)" as "(≤ *n p*)" (*at-most restriction*).

An *inclusion* of ***L*** (or over *V*) is an expression of one of the forms $u \sqsubseteq v$ or $u \sqsubseteq \neg v$, where *u* and *v* are basic concept descriptions.

An *assertion* of ***L*** (or in *V*) is an expression of one of the forms *C(a)*, ¬*C(a)*, *P(a,b)*, ¬*P(a,b)*, *(a ≈ b),* and ¬*(a ≈ b)*, where *C* is an atomic concept, *P* is an atomic role, and *a* and *b* are object names.

PUC-Rio - Certificação Digital Nº 1112681/CA

We also say that *(a ≈ b)* and ¬*(a ≈ b)* are an *equality* and an *inequality*, respectively. Note that we allow equality and inequality assertions to occur in order to capture owl:sameAs and owl:differentFrom OWL properties.

A *formula* of ***L*** (or over *V*) is an inclusion or an assertion of ***L***.

An *interpretation* ***s*** for ***L*** consists of a nonempty set Δ^s, the *domain* of ***s***, and an *interpretation function*, also denoted ***s***, with the usual definition (Artale et al., 2009). We use ***s***(*u*) to indicate the value that ***s*** assigns to an expression *u* of ***L***. We say that ***s*** *satisfies* a formula σ of ***L*** or that ***s*** is a *model* of σ, denoted $\boldsymbol{s} \models \sigma$, iff

$\boldsymbol{s}(u) \subseteq \boldsymbol{s}(v)$ if σ is of the form $u \sqsubseteq v$

$\boldsymbol{s}(u) \subseteq \boldsymbol{s}(\neg v)$ if σ is of the form $u \sqsubseteq \neg v$

$\boldsymbol{s}(a) \in \boldsymbol{s}(C)$ if σ is of the form $C(a)$

$(\boldsymbol{s}(a),\boldsymbol{s}(b)) \in \boldsymbol{s}(P)$ if σ is of the form $P(a,b)$

$\boldsymbol{s}(a) = \boldsymbol{s}(b)$ if σ is of the form $(a \approx b)$

$\boldsymbol{s} \not\models \theta$ if σ is of the form $\neg\theta$

Let Σ be a set of formulas of ***L***. We say that ***s*** *satisfies*Σ or that ***s*** is a *model* of Σ, denoted $\boldsymbol{s} \models \Sigma$, iff ***s*** satisfies all formulas in Σ. We also say that Σ *logically implies* σ, denoted $\Sigma \models \sigma$, iff any model of Σ satisfies σ. Finally, we say that Σ is *satisfiable* or *consistent* iff there is a model of Σ.

Table 1 lists the types of $DL\text{-}Lite^{N}_{core}$ inclusions that represent constraints commonly used in conceptual modeling.

Table 1. Common constraint types used in conceptual modeling.

Constraint type	**Abbreviated form**	**Unabbreviated form**	**Informal semantics**
Domain Constraint	$\exists P \sqsubseteq C$	$(\geq 1\ P) \sqsubseteq C$	Property *P* has class *C* as domain, that is, if *(a,b)* is a pair in *P*, then *a* is an individual in *C*.
Range Constraint	$\exists P^{-} \sqsubseteq C$	$(\geq 1\ P^{-}) \sqsubseteq D$	Property *P* has class *D* as range, that is, if *(a,b)* is a pair in *P*, then *b* is an individual in *D*.
minCardinality Constraint		$C \sqsubseteq (\geq n\ P)$ or $C \sqsubseteq (\geq n\ P^{-})$	Property *P* or its inverse P^{-} maps each individual in class *C* to at least *n* distinct individuals.
maxCardinality Constraint	$C \sqsubseteq (\leq n\ P)$ or $C \sqsubseteq (\leq n\ P^{-})$	$C \sqsubseteq \neg(\geq n{+}1\ P)$ or $C \sqsubseteq \neg(\geq n{+}1\ P^{-})$	Property *P* or its inverse P^{-} maps each individual in class *C* to at most *n* distinct individuals.

PUC-Rio - Certificação Digital Nº 1112681/CA

Subset Constraint		$C \sqsubseteq D$	Each individual in *C* is also in *D*, that is, class *C* denotes a subset of class *D*.
Disjointness Constraint		$C \sqsubseteq \neg D$	No individual is in both *C* and *D*, that is, classes *C* and *D* are disjoint.

In this thesis, we adopt the notion of constraint graph to capture the structure of sets of constraints. Given a set of constraints Σ and a set of concept descriptions Ω, the *constraint graph* that *represents* Σ and Ω, denoted $G(\Sigma,\Omega)$, can be constructed as detailed in Casanova et al. (2010). When Ω is the empty set, we simply write $G(\Sigma)$ and say that the graph *represents* Σ. Section 3.3 contains an example of a constraint graph.

A constraint graph is also fundamental to construct the set of constraints of a data mashup specification. In our approach, the data mashup constraints are not merely the constraints of the domain ontology, but must be those that are logical consequences of the domain ontology constraints and that involve only the selected symbols of the vocabulary of the data mashup. An example of a constraint graph is shown in the next chapter.

PUC-Rio - Certificação Digital Nº 1112681/CA

2.7 The Description Logic SROIQ and the OWL 2 Direct Semantics

The semantics of the Description Logic $\mathcal{SROIQ}$ (Horrocks and Sattler, 2006) provide the basis for the direct model-theoretical semantics of OWL 2 (OWL 2, 2012c).

Very briefly, $\mathcal{SROIQ}$ is an extension of the Description Logic underlying OWL-DL, $\mathcal{SHOIN}$, with a number of expressive means useful in practice. $\mathcal{SROIQ}$ includes complex role inclusion axioms of the form $R \circ S \sqsubseteq R$ or $S \circ R \sqsubseteq R$ to express propagation of one property along another one, which have proven useful in medical terminologies.

Furthermore, $\mathcal{SROIQ}$ extends $\mathcal{SHOIN}$ with reflexive, antisymmetric and irreflexive roles, disjoint roles, a universal role, while constructing $\exists R$.Self which allows, for instance, the definition of concepts such as a “narcist”. Finally, $\mathcal{SROIQ}$ considers negated role assertions and qualified number restrictions.

$\mathcal{SROIQ}$ has a tableau-based reasoning algorithm that combines the use of automata to keep track of universal value restrictions with the techniques specifically developed for $\mathcal{SROIQ}$ (Horrocks and Sattler, 2006).

2.8 Summary

This chapter covered two distinct sets of topics. Sections 2.1 to 2.3 presented a brief and informal introduction to data integration, query answering, and mashup applications, which are database problems directly related to this thesis. Sections 2.4 to 2.7 surveyed formal concepts used throughout the thesis: model checking, OWL2 and two dialects of Description Logics: DL-Lite Core with arbitrary number restrictions and $\mathcal{SROIQ}$. DL Lite Core with arbitrary number restrictions was covered in more detail, since it provides the basis for some of the contributions described in Chapter 5.

PUC-Rio - Certificação Digital Nº 1112681/CA

3
A Case Study of Data Mashup

In this chapter, we discuss the problem of building consistent data mashups. The explanation is based on a schematic example, in the context of Linked Data, which presents each step of this problem in an intuitive way.

3.1
Brief Introduction to Linked Data

In recent years, the Web has evolved from a global information space of linked documents to one where both documents and data are linked. This evolution, called *Linked Data*, is reflected in a set of best practices for publishing and connecting structured data on the Web (Bizer et al., 2007).

The adoption of the Linked Data best practices led to the extension of the Web with a global data space connecting data from several domains. These practices include (Bizer et al., 2009):

(i) Using URIs for properly identifying the resources;
(ii) Using technologies, such as RDF and SPARQL, respectively, for describing and querying these resources;
(iii) Reusing URIs to create links between data from different sources, in order to connect these sources and to discover additional things about data, as one navigates through these links.

Current Linked Data technologies facilitate links to be created between records in distinct databases. They have been adopted by an increasing number of Web data providers over the last years, leading to the creation of a global data space containing billions of assertions, called the *Web of Data*.

Unlike Web 2.0 mashups, defined over a fixed set of data sources, Linked Data applications may potentially discover new data sources at runtime by following links between different databases, and can thus deliver better answers, as new data sources appear on the Web.

PUC-Rio - Certificação Digital Nº 1112681/CA

3.2 Description of the Data Sources

In this chapter, we consider that data is provided by Linked Data sources, i.e., we assume that data sources are published following the Linked Data principles.

Consider a Linked Data mashup service that covers a given domain, defined by a *domain ontology* and a set of the Linked Data sources, modeled by *application ontologies*. We consider only one domain ontology for simplicity.

Furthermore, we assume that:

1. The application ontology vocabularies are subsets of the domain ontology;
2. The Linked Data mashup service has access to the vocabularies of the application ontologies (but not to their constraints);
3. The Linked Data mashup service has access to the vocabulary and constraints of the domain ontology.

These assumptions are consistent with the current Linked Data practices, which promote:

- Reuse of known vocabularies to define a Linked Data source;
- Adoption of a VoiD document to indicate the vocabularies – but not the constraints – that a Linked Data source uses;
- Adoption of repositories that provide access to the full definition – vocabulary and constraints – of commonly used domain ontologies.

We cannot assume, however, that the data retrieved from different Linked Data sources is consistent with the constraints of the domain ontology, for two reasons. First, we have no guarantee that each Linked Data source returns consistent data; in fact, we do not even know what constraints the Linked Data source respects. Second, even if each Linked Data source returned data that is consistent with the domain ontology constraints, the combined data might be inconsistent. In view of these observations, the Linked Data mashup service must always analyze the data coming from different Linked Data sources to identify and isolate inconsistent data.

PUC-Rio - Certificação Digital Nº 1112681/CA

3.3 Description of the Data Mashup

We assume that the user is responsible for describing the *data mashup vocabulary* V_M as a subset of the domain ontology vocabulary.

The set Σ_M of *mashup constraints* are those that are logical consequences of the domain ontology constraints and that involve only the symbols in V_M. The generation of such constraints is discussed in Sacramento et al., 2012.

The *mashup ontology* is the pair $\boldsymbol{O}_M = (V_M, \Sigma_M)$ and represents a conceptual model of what the user observes.

In our example, assume that:

- The mashup vocabulary V_M has four classes, A, B, C and D, and a property p.
- The set of mashup constraints Σ_M is the one shown in the second column of Table 2, using the basic notation of Description Logic:

  $\Sigma_M = \{ (\geq 1\ p^{-}) \sqsubseteq D, C \sqsubseteq (\geq 1\ p), C \sqsubseteq \neg(\geq 2\ p), A \sqsubseteq C, B \sqsubseteq C, A \sqsubseteq \neg B \}$

Figure 1 depicts the constraint graph $G(\Sigma_M)$ that represents Σ_M, which is constructed as follows.

We say that the *complement* of a basic concept description e is $\neg e$, and vice-versa. If c is a concept description, then $\bar{c}$ denotes the complement of c. The nodes of $G(\Sigma_M)$ are labeled with expressions and their complements (for simplicity, we say "node u" instead of "node labeled with u").

For each inclusion $u \sqsubseteq v$ in Σ_M, there are nodes in $G(\Sigma_M)$ labeled with u, $\bar{u}$, v and $\bar{v}$, and arcs from node u to node v and from node $\bar{v}$ to node $\bar{u}$. For example, the constraint $A \sqsubseteq \neg B$ generates two arcs: an arc from node A to node $\neg B$ and an arc from node B to node $\neg A$.

$G(\Sigma_M)$ is such that, if there is a path from node u to node v, then Σ_M logically implies $u \sqsubseteq v$. For example, in constraint σ_3, since there is a path from node C to node $\neg(\geq 2\ p)$, Σ_M logically implies $C \sqsubseteq \neg(\geq 2\ p)$. Furthermore, since there is a path from node B to node C and a path from node C to node $\neg(\geq 2\ p)$, Σ_M logically implies $B \sqsubseteq \neg(\geq 2\ p)$.

PUC-Rio - Certificação Digital Nº 1112681/CA

Table 2. Set of mashup constraints Σ_M.

#	*Constraint*	*Informal specification*
σ_1	$(\geq 1\ p^{-}) \sqsubseteq D$	The range of p is D.
σ_2	$C \sqsubseteq (\geq 1\ p)$	C is contained in the domain of p.
σ_3	$C \sqsubseteq \neg(\geq 2\ p)$	p associates at most one individual to every individual in C.
σ_4	$A \sqsubseteq C$	A is a subset of C.
σ_5	$B \sqsubseteq C$	B is a subset of C.
σ_6	$A \sqsubseteq \neg B$	A is disjoint from B.

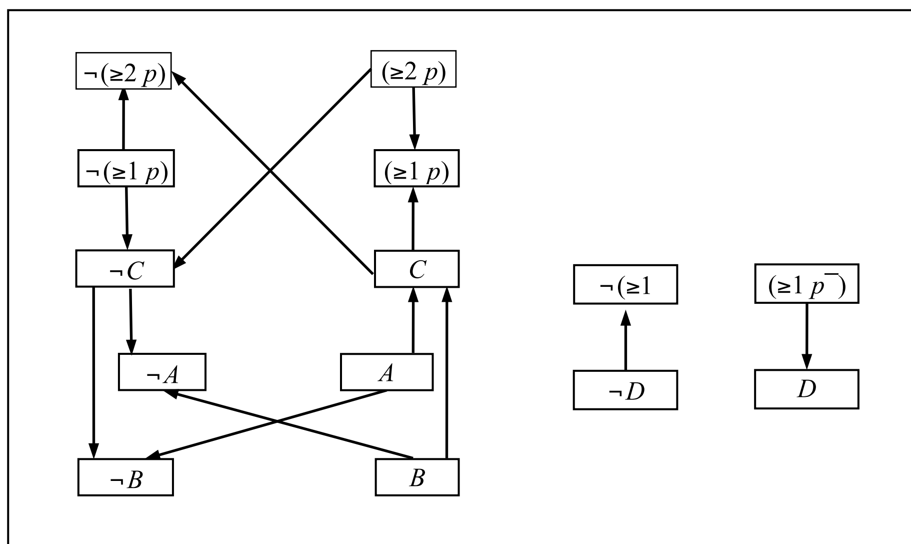

Figure 1. The constraint graph $G(\Sigma_M)$ for Σ_M.

PUC-Rio - Certificação Digital Nº 1112681/CA

Table 3. Data sources and matching rules.

Source	Vocabulary	#	Matching Rule
S1	S1:B S1:D S1:p	m_1 m_2 m_3	$B(x) \leftarrow S1{:}B(x)$ $D(x) \leftarrow S1{:}D(x)$ $p(x,y) \leftarrow S1{:}\ p(x,y)$
S2	S2:A S2:D S2:p	m_4 m_5 m_6	$A(x) \leftarrow S2{:}A(x)$ $D(x) \leftarrow S2{:}D(x)$ $p(x,y) \leftarrow S2{:}\ p(x,y)$
S3	S3:C	m_7	$C(x) \leftarrow S3{:}C(x)$

Table 4. Assertions expressing data from sources *S1*, *S2* and *S3*.

S1			S2			S3		
#	Original	Trans	#	Original	Trans	#	Original	Trans
S1.1	S1:B(a)	B(a)	S2.1	S2:A(a)	A(a)	S3.1	S3:C(d)	C(d)
S1.2	S1:D(b)	D(b)	S2.2	S2:D(c)	D(c)			
S1.3	S1:p(a,b)	p(a,b)	S2.3	S2:p(a,c)	p(a,c)			
			S2.4	S2:p(d,c)	p(d,c)			

Table 5. Maximal consistent subsets of *S*.

M1	$B(a)$, $C(a)$, $\neg A(a)$, $p(a,b)$, $p(d,c)$, $(\geq 1\ p)(a)$, $(\geq 1\ p^-)(b)$, $(\geq 1\ p^-)(c)$, $\neg(\geq 2\ p)(a)$, $D(b)$, $D(c)$, $C(d)$, $(\geq 1\ p)(d)$, $\neg(\geq 2\ p)(d)$
M2	$B(a)$, $C(a)$, $\neg A(a)$, $p(a,c)$, $p(d,c)$, $(\geq 1\ p)(a)$, $(\geq 1\ p^-)(c)$, $\neg(\geq 2\ p)(a)$, $D(b)$, $D(c)$, $C(d)$, $(\geq 1\ p)(d)$, $\neg(\geq 2\ p)(d)$
M3	$A(a)$, $C(a)$, $\neg B(a)$, $p(a,b)$, $p(d,c)$, $(\geq 1\ p)(a)$, $(\geq 1\ p^-)(b)$, $(\geq 1\ p^-)(c)$, $\neg(\geq 2\ p)(a)$, $D(b)$, $D(c)$, $C(d)$, $(\geq 1\ p)(d)$, $\neg(\geq 2\ p)(d)$
M4	$A(a)$, $C(a)$, $\neg B(a)$, $p(a,c)$, $p(d,c)$, $(\geq 1\ p)(a)$, $(\geq 1\ p^-)(c)$, $\neg(\geq 2\ p)(a)$, $D(b)$, $D(c)$, $C(d)$, $(\geq 1\ p)(d)$, $\neg(\geq 2\ p)(d)$
M5	$\boldsymbol{\neg C(a)}$, $\neg B(a)$, $\neg A(a)$, $p(a,b)$, $p(a,c)$, $p(d,c)$, $(\geq 1\ p^-)(b)$, $(\geq 1\ p^-)(c)$, $(\geq 2\ p)(a)$, $D(b)$, $D(c)$, $C(d)$, $(\geq 1\ p)(d)$, $\neg(\geq 2\ p)(d)$

3.4 Registering the Linked Data Sources

The process of registering a Linked Data source with a data mashup service involves matching the vocabulary of the Linked Data source with the vocabulary of the domain ontology. The matching step is outside the scope of this thesis, but it is a trivial process in our approach, as the vocabularies of the Linked Data sources are subsets of the vocabulary of the domain ontology.

In our example, assume that the data mashup service has access to three Linked Data sources, *S1*, *S2* and *S3*, which vocabularies are shown in the second

PUC-Rio - Certificação Digital Nº 1112681/CA

column of Table 3. Also assume that the vocabularies of the Linked Data sources are matched with the vocabulary of the domain ontology, as shown in the last column of Table 3, through a set of *matching rules*.

We express these matching rules using a Datalog-like notation. For simplicity, for the classes that match, we used the same name as in the domain ontology, prefixed with "*Si:*". For example, the matching rule m_1: $B(x) \leftarrow S1{:}B(x)$ shown in line 1 of Table 3 tries to match instances *x* in class *B* from *S1* with instances *x* of class *B* in the domain ontology. This kind of matching rule is very simple and can be automatically generated.

3.5 Querying the Linked Data Sources and Populating the Linked Data Mashup

Then, the mashup service prepares a set of queries over the data sources to create the data mashup to be shown to the user. Such queries are, in fact, reinterpretations of the matching rules, where the body of a rule is the *where clause* of the query and the head is the *target clause*.

For the sake of argument, assume that the queries return the assertions shown in the columns labeled "***Original***" in Table 4. Then, we translate the query results to assertions over the vocabulary of the domain ontology with the help of the matching rules, as shown in the columns labeled "***Trans***" ***(Translated)*** in Table 4.

The key problem we address in this thesis is how to design a mashup service that creates a maximal subset ***M*** of the set of assertions collected from the data sources, in such a way that ***M*** is consistent with the constraints of the mashup ontology; if there is more than one subset, the service should offer the user the opportunity to browse all such subsets.

Let us first discuss why this is a problem. First, observe that Table 4 shows all assertions obtained from the data sources, after applying the matching rules.

Example 1: Let ***A*** be the set of all such assertions:

$$\boldsymbol{A} = \{B(a), D(b), p(a,b), A(a), D(c), p(a,c), p(d,c), C(d)\}$$

Recall that the set of constraints is:

$$\Sigma_M = \{ (\geq 1\ p^{-}) \sqsubseteq D,\ C \sqsubseteq (\geq 1\ p),\ C \sqsubseteq \neg(\geq 2\ p),\ A \sqsubseteq C,\ B \sqsubseteq C,\ A \sqsubseteq \neg B \}$$

PUC-Rio - Certificação Digital Nº 1112681/CA

Note that assertion *A(a)* alone (or *B(a)*) violates constraint $A \sqsubseteq C$ (or constraint $B \sqsubseteq C$), since there is no assertion *C(a)*. To avoid constraint violations such as this, we allow the creation of new derived assertions. For example, we generate assertion *C(a)* whenever assertion *A(a)* is present. Furthermore, an assertion such as *p(a,b)* induces assertions $(\geq 1\ p)(a)$ and $(\geq 1\ p^-)(b)$, by definition of at-least restriction, which we must also take into account.

Let $\boldsymbol{S}$ be the set of assertions obtained by applying such strategy to the assertions in $\boldsymbol{A}$. We construct $\boldsymbol{S}$ as follows.

First, we initialize $\boldsymbol{S_0}$ with $\boldsymbol{A}$:

$\boldsymbol{S_0} = \boldsymbol{A} = \{B(a), D(b), p(a,b), A(a), D(c), p(a,c), p(d,c), C(d)\}$

Then, from $\boldsymbol{S_0}$ and Σ_M, we obtain new assertions:

- *C(a)*, from *B(a)* and constraint $B \sqsubseteq C$
- $\neg A(a)$, from *B(a)* and constraint $A \sqsubseteq \neg B$ (which is equivalent to $B \sqsubseteq \neg A$)
- $(\geq 1\ p^-)(b)$, from *p(a,b)* and the definition of $(\geq 1\ p^-)$
- *C(a)*, from *A(a)* and constraint $A \sqsubseteq C$
- $\neg B(a)$, from *A(a)* and constraint $A \sqsubseteq \neg B$
- $(\geq 1\ p^-)(c)$, from *p(a,c)* and the definition of $(\geq 1\ p^-)$
- $(\geq 2\ p)(a)$, from *p(a,b), p(a,c)* and the definition of $(\geq 2\ p)$
- $(\geq 1\ p^-)(c)$, from *p(d,c)* and the definition of $(\geq 1\ p^-)$
- $(\geq 1\ p)(d)$, from *C(d)* and constraint $C \sqsubseteq (\geq 1\ p)$
- $\neg(\geq 2\ p)(d)$, from *C(d)* and constraint $C \sqsubseteq \neg(\geq 2\ p)$

The new set of assertions is:

$\boldsymbol{S_1} = \boldsymbol{S_0} \cup \{C(a), \neg A(a), (\geq 1\ p^-)(b), \neg B(a), (\geq 1\ p^-)(c), (\geq 2\ p)(a),$
$(\geq 1\ p)(d), \neg(\geq 2\ p)(d)\}$

Now, from $\boldsymbol{S_1}$ and Σ_M, we obtain new assertions:

- $(\geq 1\ p)(a)$, from *C(a)* and constraint $C \sqsubseteq (\geq 1\ p)$
- $\neg(\geq 2\ p)(a)$, from *C(a)* and constraint $C \sqsubseteq \neg(\geq 2\ p)$
- *D(b)*, from $(\geq 1\ p^-)(b)$ and constraint $(\geq 1\ p^-) \sqsubseteq D$
- *D(c)*, from $(\geq 1\ p^-)(c)$ and constraint $(\geq 1\ p^-) \sqsubseteq D$
- $\neg C(a)$, from $(\geq 2\ p)(a)$ and constraint $C \sqsubseteq \neg(\geq 2\ p)$ (which is equivalent to $(\geq 2\ p) \sqsubseteq \neg C$)

PUC-Rio - Certificação Digital Nº 1112681/CA

The new set of assertions is:

$\boldsymbol{S_2} = \boldsymbol{S_1} \cup \{(\geq 1\ p)(a),\ \neg(\geq 2\ p)(a),\ \neg C(a)\}$

No new assertions can be obtained from the assertions added to $\boldsymbol{S_1}$ to create $\boldsymbol{S_2}$. Hence, the final set of assertions $\boldsymbol{S}$ is:

$\boldsymbol{S} = \{\ B(a),\ D(b),\ p(a,b),\ A(a),\ D(c),\ p(a,c),\ p(d,c),\ C(d),$
$C(a),\ \neg A(a),\ (\geq 1\ p^{-})(b),\ \neg B(a),\ (\geq 1\ p^{-})(c),\ (\geq 2\ p)(a),$
$(\geq 1\ p)(d),\ \neg(\geq 2\ p)(d),\ (\geq 1\ p)(a),\ \neg(\geq 2\ p)(a),\ \neg C(a)\}$

or, reordering assertions lexicographically:

$\boldsymbol{S} = \{\ A(a),\ \neg A(a),$
$B(a),\ \neg B(a),$
$C(a),\ \neg C(a),\ C(d),$
$D(b),\ D(c),$
$p(a,b),\ p(a,c),\ p(d,c),$
$(\geq 1\ p)(a),\ (\geq 1\ p)(d),\ (\geq 1\ p^{-})(b),\ (\geq 1\ p^{-})(c),$
$(\geq 2\ p)(a),\ \neg(\geq 2\ p)(a),\ \neg(\geq 2\ p)(d)\ \}$

Suppose that the mashup service passes $\boldsymbol{S}$ directly to the user. The service would then be passing inconsistent data, since the following pairs of assertions are inconsistent:

(1) $A(a),\ \neg A(a)$

(2) $B(a),\ \neg B(a)$

(3) $C(a),\ \neg C(a)$

(4) $(\geq 2\ p)(a),\ \neg(\geq 2\ p)(a)$

In fact, there are five maximal consistent subsets of $\boldsymbol{S}$, as shown in Table 5. The construction of the mashups in Table 5 requires some explanation, though.

First, we must properly allocate each pair of assertions presented in (1), (2), (3) and (4) to distinct mashups, since they are inconsistent, as they represent contradictory information.

Second, we must clearly allocate assertions $A(a)$ and $B(a)$ to distinct mashups, since they violate the disjointness constraint $A \sqsubseteq \neg B$.

Third, we must allocate assertion $C(a)$ and each of the two assertions, $p(a,b)$ and $p(a,c)$, to distinct mashups, since the three assertions together violate the maxCardinality constraint $C \sqsubseteq \neg(\geq 2\ p)$.

PUC-Rio - Certificação Digital Nº 1112681/CA

Hence, we create four consistent mashups ***M1***, ***M2***, ***M3*** and ***M4***. However, and less intuitively, we may create a consistent mashup ***M5*** with two assertions, *p(a,b)* and *p(a,c)*, if we ignore assertion *C(a)*.

The negated assertions may also be exhibited to the user in order to reinforce what constraints $C \sqsubseteq \neg(\geq 2\ p)$ and $A \sqsubseteq \neg B$ impose. Consider $C \sqsubseteq \neg(\geq 2\ p)$ first. In ***M1*** and ***M2***, the negated assertions $\neg(\geq 2\ p)(a)$ and $\neg(\geq 2\ p)(d)$ indicate that p associates just one individual with instances a and d, respectively, and likewise for ***M3*** and ***M4***. In ***M5***, $\neg C(a)$ indicates that instance a cannot be considered as a member of C. Similar observations apply when we consider $A \sqsubseteq \neg B$, regarding $\neg A(a)$ and $\neg B(a)$.

3.6 Summary

The discussion in this chapter raises several questions:

Q1. How to compute the mashup constraints from the domain ontology constraints?

Q2. How to match the data source vocabularies with the vocabulary of the domain ontology?

Q3. How to derive new assertions from those obtained from the data sources (after translation) and the mashup constraints?

Q4. How to create a (maximal) consistent subset ***M*** of the set of the assertions collected from the data sources in such a way that ***M*** is consistent with the mashup constraints?

Question Q1 was discussed in Casanova et al., 2011 and Sacramento et al., 2012. Question Q2 was discussed in Sacramento et al., 2010. The matching step is in fact a trivial process, as we assumed that the vocabularies of the data sources are subsets of the vocabulary of the domain ontology.

This thesis proposes a formal approach to Questions Q3 and Q4 and, in special, it addresses Question Q4 based on the use of Default Theories.

PUC-Rio - Certificação Digital Nº 1112681/CA

4
Mashups as Default Theories

In this chapter, we first present the basic concepts of the Default Logic adopted in this thesis. Then, we formalize the problem of building consistent data mashups. Next, we present our case study, reformulated using the concepts of defaults. Finally, we prove that the mapping of the problem of building consistent data mashups from the context of Description Logics to the context of Default Logic is correct and complete, i.e., we prove that both approaches generate the same maximal consistent subsets of assertions.

PUC-Rio - Certificação Digital Nº 1112681/CA

4.1
Default Logic

Default Logic is a non-monotonic logic proposed to formalize reasoning with default assumptions (Levesque et al., 2004; Reiter, 1980). This logic allows expressing facts such as "by default, something is true".

The following definitions are formulated in the context of first-order logic, but they equally apply to any of the descriptions logics mentioned in Chapter 2.

For compatibility with the description logics of Chapter 2, we define an *assertion* as a ground atomic formula.

Let V be a first-order alphabet. A *default* over V is an expression of the form "$\varphi : \psi_1,...,\psi_n / \chi$" such that φ, $\psi_1,...,\psi_n$ and χ are closed predicate logic formulae over V and $n > 0$. The formula φ is the *prerequisite* (*pre*), $\psi_1...\psi_n$ are the *justifications* (*just*), and χ is the *consequent* (*cons*) of the default. Intuitively, the default "$\varphi : \psi_1,...,\psi_n / \chi$" means that "if φ is known and if it is consistent to assume $\psi_1,...,\psi_n$, then conclude χ". If this is the case, we say that the default was *fired*.

A *default theory* is a triple $\boldsymbol{T} = (V, \Sigma, \Delta)$, where V is a first-order alphabet, Σ is a set of first-order formulae over V, called the *axioms* of $\boldsymbol{T}$, and Δ is a countable set of defaults over V.

Let $\boldsymbol{T} = (V, \Sigma, \Delta)$ be a default theory. A set $\boldsymbol{\Gamma}$ of closed predicate logic formulae over V is an *extension* of $\boldsymbol{T}$ iff, for every closed predicate logic formula π over V, $\pi \in \boldsymbol{\Gamma}$ iff $\Sigma \cup \Theta \vDash \pi$, where

$$\Theta = \{ \chi \mid \text{“}\varphi : \psi_1, \ldots, \psi_n / \chi\text{”} \in \Delta \text{ and } \varphi \in \boldsymbol{\Gamma} \text{ and } \neg(\psi_1 \wedge \ldots \wedge \psi_n) \notin \boldsymbol{\Gamma} \}$$

An extension $\boldsymbol{\Gamma}$ of a default theory $\boldsymbol{T} = (V, \Sigma, \Delta)$ is therefore a maximal consistent set of formulae that can be derived from the axioms in $\boldsymbol{\Sigma}$ and a maximal set of defaults in Δ that can be fired in the extension (without producing inconsistencies). Generally speaking, extensions represent maximal possible world views that are based on the given default theory and that extend the underlying knowledge base (the facts that are known for sure) with plausible conjectures based on the defaults.

In this thesis, we adopt a very restricted form of default that is sufficient for formalizing the concept of a consistent data mashup.

Let V be a vocabulary, as defined in Section 2.4. A *simple default* over V is an expression of the form “$: \delta / \delta$” such that δ is an assertion over V. Hence, a simple default has no pre-requisite and the justification and the consequent are the same assertion δ. The informal interpretation of “$: \delta / \delta$” is that we can assume δ if we do not have $\neg\delta$; if this is the case, recall we said that the default was fired.

We restrict our attention to simple defaults, as reasoning in this setting is decidable in general, when the underlying Default Logic is also decidable (Baader and Hollunder, 1995). We stress that such defaults are sufficient for the purposes of formalizing mashups.

PUC-Rio - Certificação Digital Nº 1112681/CA

4.2 Mashup Default Theories

In this section, we show how to translate an ontology and a set of assertions to a default theory. In Section 4.4, we will prove that the extensions of the default theory induce consistent data mashups.

Let $\boldsymbol{O} = (V, \Sigma)$ be an ontology, where V is a vocabulary and $\boldsymbol{\Sigma}$ is a set of constraints (the underlying logic is not relevant at this point). Let $\boldsymbol{A}$ be a finite set of *assertions*.

Definition 1:

i) The *translation* of an assertion δ in $\boldsymbol{A}$ is the simple default "$: \delta / \delta$".

ii) The *mashup default theory corresponding to* $\boldsymbol{O}$ *and* $\boldsymbol{A}$ is the default theory $\boldsymbol{T} = (V, \Sigma, \Delta)$ such that V is the vocabulary of $\boldsymbol{O}$, Σ is the set of constraints of $\boldsymbol{O}$, and Δ is the finite set of simple defaults over V that translate the assertions in $\boldsymbol{A}$.

iii) An *extension* $\boldsymbol{\Gamma}$ of a mashup default theory $\boldsymbol{T} = (V, \Sigma, \Delta)$ is a set $\boldsymbol{\Gamma}$ of assertions and inclusions over V such that $\pi \in \boldsymbol{\Gamma}$ iff π is a logical consequence of $\Sigma \cup \Theta$, where $\Theta = \{ \delta / \text{"}: \delta / \delta\text{"} \in \Delta \text{ and } \neg\delta \notin \boldsymbol{\Gamma} \}$.

iv) A *consistent data mashup* for $\boldsymbol{T} = (V, \Sigma, \Delta)$ is an extension of $\boldsymbol{T}$. □

Note that simple defaults in Definition 1 (i) are ground, as the assertions in $\boldsymbol{A}$ that originate them are ground formulae. Also note that the notion of extension in Definition 1 (iii) is just a restatement of the general notion of extension, introduced in Section 4.1.

Intuitively, a set of assertions in $\boldsymbol{A}$ retrieved from the data sources will be considered together iff the corresponding simple defaults in Δ can be fired in the presence of the constraints in Σ. Furthermore, one should consider only maximal sets of such defaults to maximize the data retrieved from the data sources and shown to the user, without running into inconsistencies. But this is exactly the notion of extension, as further discussed in Section 4.4.

4.3 Case Study using Mashup Defaults

This section illustrates the concepts introduced in Definition 1.

Consider the same ontology as in Chapter 3. In particular, recall that the set of constraints Σ was given in Table 2.

Example 2: Let $\boldsymbol{A}$ be the set of all assertions obtained from the data sources, after applying a set of matching rules. In this section, we suppose, for simplicity, that $\boldsymbol{A}$ is the following set of assertions:

$$\boldsymbol{A} = \{B(a), p(a,b), A(a), p(a,c)\}$$

From the set of assertions that are logical consequences of $\boldsymbol{A}$ and from the set of constraints Σ, we obtain five (maximal) consistent subsets $\boldsymbol{M}$ of assertions, presented in Table 6 (which replaces Table 5 for the purposes of this example).

PUC-Rio - Certificação Digital Nº 1112681/CA

Table 6. Maximal consistent subsets of assertions.

M1	$B(a)$, $C(a)$, $\neg A(a)$, $p(a,b)$, $(\geq 1\ p)(a)$, $(\geq 1\ p^{-})(b)$, $\neg(\geq 2\ p)(a)$
M2	$B(a)$, $C(a)$, $\neg A(a)$, $p(a,c)$, $(\geq 1\ p)(a)$, $(\geq 1\ p^{-})(c)$, $\neg(\geq 2\ p)(a)$
M3	$A(a)$, $C(a)$, $\neg B(a)$, $p(a,b)$, $(\geq 1\ p)(a)$, $(\geq 1\ p^{-})(b)$, $\neg(\geq 2\ p)(a)$
M4	$A(a)$, $C(a)$, $\neg B(a)$, $p(a,c)$, $(\geq 1\ p)(a)$, $(\geq 1\ p^{-})(c)$, $\neg(\geq 2\ p)(a)$
M5	$\neg C(a)$, $\neg B(a)$, $\neg A(a)$, $p(a,b)$, $p(a,c)$, $(\geq 1\ p^{-})(b)$, $(\geq 1\ p^{-})(c)$, $(\geq 2\ p)(a)$

Now, we adapt our case study to default theories.

We first map the ontology $\boldsymbol{O} = (V, \Sigma)$ and the assertions $\boldsymbol{A}$ into a mashup default theory $\boldsymbol{T} = (V, \Sigma, \Delta)$, as in Definition 1, and consider a consistent data mashup as an extension of this theory. Each assertion in $\boldsymbol{A}$ is translated into a simple default, generating the set of defaults $\Delta = \{ \delta_1 , \delta_2 , \delta_3 , \delta_4\}$, where:

δ_1 = ": $B(a)$ / $B(a)$" δ_2 = ": $p(a,b)$ / $p(a,b)$"

δ_3 = ": $A(a)$ / $A(a)$" δ_4 = ": $p(a,c)$ / $p(a,c)$"

Table 7 presents the five extensions of the default theory $\boldsymbol{T}$, which in fact correspond to the five (maximal) consistent subsets of assertions shown in Table 6. For simplicity, we omit from the extension the derived inclusions, that is, we list only the assertions. In the last column of Table 7, we indicate which defaults were fired to obtain the extension. For example, to obtain extension $\boldsymbol{\Gamma_1}$, default δ_1 = ": $B(a)$ / $B(a)$" was fired to generate assertion $B(a)$, and default δ_2 = ": $p(a,b)$ / $p(a,b)$" was fired to generate assertion $p(a,b)$. The other assertions of $\boldsymbol{\Gamma_1}$ were generated considering the set of constraints Σ.

Table 7. Assertions in the extensions of the default theory *T*.

	Assertions in the extension	***Defaults fired***
$\boldsymbol{\Gamma_1}$	$B(a)$, $C(a)$, $\neg A(a)$, $p(a,b)$, $(\geq 1\ p)(a)$, $(\geq 1\ p^{-})(b)$, $\neg(\geq 2\ p)(a)$	δ_1, δ_2
$\boldsymbol{\Gamma_2}$	$B(a)$, $C(a)$, $\neg A(a)$, $p(a,c)$, $(\geq 1\ p)(a)$, $(\geq 1\ p^{-})(c)$, $\neg(\geq 2\ p)(a)$	δ_1, δ_4
$\boldsymbol{\Gamma_3}$	$A(a)$, $C(a)$, $\neg B(a)$, $p(a,b)$, $(\geq 1\ p)(a)$, $(\geq 1\ p^{-})(b)$, $\neg(\geq 2\ p)(a)$	δ_3, δ_2
$\boldsymbol{\Gamma_4}$	$A(a)$, $C(a)$, $\neg B(a)$, $p(a,c)$, $(\geq 1\ p)(a)$, $(\geq 1\ p^{-})(c)$, $\neg(\geq 2\ p)(a)$	δ_3, δ_4
$\boldsymbol{\Gamma_5}$	$\neg C(a)$, $\neg B(a)$, $\neg A(a)$, $p(a,b)$, $p(a,c)$, $(\geq 1\ p^{-})(b)$, $(\geq 1\ p^{-})(c)$, $(\geq 2\ p)(a)$	δ_2, δ_4

PUC-Rio - Certificação Digital Nº 1112681/CA

If we fired more defaults in each extension, we would generate inconsistencies. So, the assertions that belong to each extension $\boldsymbol{\Gamma_i}$ are those that are logical consequence of the constraints of Σ and the consequents of a subset of the defaults in Δ, and that represent maximal consistent subsets. In the next chapter, we will explain how to compute such extensions.

4.4
Correctness of the Default Model for Mashups

Consider the following problem, which we call *Ontology Mashup Problem*:

Instance: An ontology $\boldsymbol{O} = (V, \Sigma)$ and be a finite set $\boldsymbol{A}$ of assertions over V.

Question: Find a maximal set of assertions $\boldsymbol{M}$ such that $\boldsymbol{M} \subseteq \boldsymbol{A}$ and $\Sigma \cup \boldsymbol{M}$ is consistent. We say that $\boldsymbol{M}$ is an *answer* to the problem defined by $\boldsymbol{O}$ and $\boldsymbol{A}$.

The set $\boldsymbol{A}$ models data obtained from the data sources to populate the classes and properties in V. Note that $\Sigma \cup \boldsymbol{A}$ may not be satisfiable, as illustrated in Chapter 3. The set of assertions derivable from $\Sigma \cup \boldsymbol{M}$ (and not just the assertions in $\boldsymbol{M}$) represents the data that the user observes. Note that $\boldsymbol{M}$ is always a maximal subset of $\boldsymbol{A}$ such that $\Sigma \cup \boldsymbol{M}$ is consistent.

In particular, if $\boldsymbol{O} = (V, \Sigma)$ is a $DL\text{-}Lite^{\mathcal{N}}_{core}$ ontology, we can prove that the Ontology Mashup Problem is NP-Complete by a reduction of the satisfiability problem of $DL\text{-}Lite^{\mathcal{N}}_{core}$ knowledge bases with equality and inequality constraints (Artale et al., 2009).

We are now in a position to reformulate the Ontology Mashup Problem as the *Default Logic Mashup Problem*:

Instance: The mashup default theory $\boldsymbol{T} = (V, \Sigma, \Delta)$ corresponding to an ontology $\boldsymbol{O} = (V, \Sigma)$ and a finite set $\boldsymbol{A}$ of assertions over V.

Question: Find an extension $\boldsymbol{\Gamma}$ of $\boldsymbol{T}$. We say that $\boldsymbol{\Gamma}$ is an *answer* to the problem defined by $\boldsymbol{T}$.

In the rest of this section, we prove that we can map an instance of the Ontology Mashup problem into an equivalent instance of the Default Logic Mashup problem, which establishes the correctness of the default model for mashups.

PUC-Rio - Certificação Digital Nº 1112681/CA

Lemma 1:

Let $\boldsymbol{O} = (V, \Sigma)$ be an ontology and $\boldsymbol{A}$ be the finite set of assertions over V. Let $\boldsymbol{T} = (V, \Sigma, \Delta)$ be the mashup default theory corresponding to $\boldsymbol{O}$ and $\boldsymbol{A}$.

i) Let $\boldsymbol{M}$ be an answer to the problem defined by $\boldsymbol{O}$ and $\boldsymbol{A}$. Let $\boldsymbol{\Gamma}$ be the set of assertions and inclusions over V that are logical consequence of $\Sigma \cup \boldsymbol{M}$. Then, $\boldsymbol{\Gamma}$ is an extension of $\boldsymbol{T}$.

ii) Let $\boldsymbol{\Gamma}$ be an extension of $\boldsymbol{T}$. Let $\boldsymbol{M} \subseteq \boldsymbol{A}$ be the set of assertions δ such that "$: \delta / \delta$" was fired in $\boldsymbol{\Gamma}$. Then, $\boldsymbol{M}$ is an answer to the problem defined by $\boldsymbol{O}$ and $\boldsymbol{A}$.

Proof:

Part (i).

Let $\boldsymbol{M}$ be an answer to the problem defined by $\boldsymbol{O}$ and $\boldsymbol{A}$. Then, we have:

(1) $\boldsymbol{M}$ is a maximal set of assertions such that $\boldsymbol{M} \subseteq \boldsymbol{A}$ and $\Sigma \cup \boldsymbol{M}$ is consistent.

Define $\boldsymbol{\Gamma}$ as the following set of assertions and inclusions over V:

(2) $\pi \in \boldsymbol{\Gamma}$ iff π is a logical consequence of $\Sigma \cup \boldsymbol{M}$.

Define Θ as the following set of assertions over V:

(3) $\Theta = \{ \delta \,/\, \text{“}: \delta / \delta\text{”} \in \Delta \textit{ and } \neg\delta \notin \boldsymbol{\Gamma} \}$

By (1), $\Sigma \cup \boldsymbol{M}$ is consistent. Hence, by (2), we have:

(4) For each assertion $\delta \in \boldsymbol{A}$, if $\delta \in \boldsymbol{M}$ then $\neg\delta \notin \boldsymbol{\Gamma}$.

By (1), $\boldsymbol{M}$ is maximal. Hence, by (2), we have:

(5) For each assertion $\delta \in \boldsymbol{A}$, if $\neg\delta \notin \boldsymbol{\Gamma}$ then $\delta \in \boldsymbol{M}$.

Indeed, assume that there is an assertion $\delta' \in \boldsymbol{A}$ such that $\neg\delta' \notin \boldsymbol{\Gamma}$ and $\delta' \notin \boldsymbol{M}$. Then, since $\neg\delta' \notin \boldsymbol{\Gamma}$, we define $\boldsymbol{M'} = \boldsymbol{M} \cup \{\delta'\}$ so that $\boldsymbol{\Gamma} \cup \boldsymbol{M'}$ would be consistent. Since $\boldsymbol{M} \subseteq \boldsymbol{M'} \subseteq \boldsymbol{A}$, we have that $\boldsymbol{M}$ is not maximal, which contradicts (1) Hence, (5) holds.

PUC-Rio - Certificação Digital Nº 1112681/CA

Then, from (3), (4) and (5), and the definition of Δ, we have:

(6) $\Theta = \{\ \delta\ /\ \text{``}:\ \delta\ /\ \delta\text{''} \in \Delta\ \textit{and}\ \neg\delta \notin \boldsymbol{\Gamma}\ \} = \{\ \delta\ /\ \text{``}:\ \delta\ /\ \delta\text{''} \in \Delta\ \textit{and}\ \delta \in \boldsymbol{M}\ \} = \boldsymbol{M}$

Hence, by (2) and (6), we have:

(7) $\pi \in \boldsymbol{\Gamma}$ iff π is a logical consequence of $\Sigma \cup \Theta$.

That is, $\boldsymbol{\Gamma}$ is an extension of $\boldsymbol{T}$.

Part (ii).

Let $\boldsymbol{\Gamma}$ be an extension of $\boldsymbol{T}$. Then, by Definition 1, we have:

(8) $\pi \in \boldsymbol{\Gamma}$ iff π is a logical consequence of $\Sigma \cup \Theta$,
where $\Theta = \{\ \delta\ /\ \text{``}:\ \delta\ /\ \delta\text{''} \in \Delta\ \textit{and}\ \neg\delta \notin \boldsymbol{\Gamma}\ \}$

Then, by reversing the argument in Part (i), we can show that Θ is an answer to the problem defined by $\boldsymbol{O}$ and $\boldsymbol{A}$. □

PUC-Rio - Certificação Digital Nº 1112681/CA

Lemma 1, indeed, establishes that we can map an instance of the Ontology Mashup problem into an equivalent instance of the Default Logic Mashup problem.

4.5 Summary

Recall from Section 3.8 the following question:

Q4. How to create a (maximal) consistent subset $\boldsymbol{M}$ of the set of the assertions collected from the data sources in such a way that $\boldsymbol{M}$ is consistent with the mashup constraints?

In this chapter, we discussed how to rephrase this question as the central problem of Default Logic, viz., how to compute extensions. Repeating the argument just after Definition 1, a set of assertions in $\boldsymbol{A}$ retrieved from the data sources will be considered together iff the corresponding simple defaults can be fired in the presence of the constraints. Furthermore, one should consider only

maximal sets of such defaults to maximize the data retrieved from the data sources and shown to the user, without running into inconsistencies.

PUC-Rio - Certificação Digital Nº 1112681/CA

5 Computing Consistent Data Mashups

In this chapter, we first describe methods to compute consistent data mashups: a brute force method; an operational method, called *process trees*; and a new method, called *mashup default trees*. Then, we discuss how to test the consistency of a positive set of assertions against an ontology and present an optimized procedure for $DL\text{-}Lite^{\mathcal{N}}_{core}$ ontologies. Finally, we present heuristics to improve the performance of the mashup default tree method.

5.1 A Brute Force Method to Compute Extensions

PUC-Rio - Certificação Digital Nº 1112681/CA

A *brutel force* or *exhaustive* method consists of systematically enumerating all candidate solutions of a problem and checking whether each candidate solution is indeed a solution for the problem. Although brute force methods are simpler to implement and they are usually capable of finding a solution; if it exists, their cost is proportional to the number of candidate solutions that, in many practical problems, tends to grow very quickly as the size of the problem increases. Therefore, such methods should be typically used when the problem size is limited, or when the simplicity of the implementation is more relevant than the performance that must be achieved.

In this section, we introduce a brute force method to compute extensions of a default theory $\boldsymbol{T} = (V, \Sigma, \Delta)$, under the assumption that Δ is a finite set of generic defaults. Section 5.3 will discuss how to compute extensions of mashup default theories, i.e, default theories with simple defaults only.

We note that, for generic defaults, the order in which the defaults are fired do matter. Therefore, the brute force method has to test sequences of defaults, and not just sets. For example, consider the following defaults:

1. $: A / A$
2. $A : B / B$
3. $B : C / C$

Then, such defaults can only be fired in the above order to generate a maximal extension, since one default makes the pre-condition of the next true.

Note that, a candidate solution, in this case, is defined by a sequence of defaults in Δ. Therefore, if there are n defaults in Δ, the number of candidate solutions is $C = (n! / (n - 1)!) + (n! / (n - 2)!) + \ldots + (n! / (n - n)!) + 1$, that is, the number of subsequences of elements of Δ. Therefore, the complexity of the force brute method is at maximum factorial, i.e., $O(n!)$, provided that we can check the consistency of firing a finite set of generic defaults in the presence of a set of constraints in exponential time.

Figure 2 shows, in pseudo code, the **Compute_Brute_Force** procedure**,** a simple brute-force algorithm that generates and examines all candidate solutions in a systematic and exhaustive manner and returns only the maximal ones.

Compute_Brute_Force uses the **Test-Consistency** procedure, which tests the consistency of a set of assertions against a predefined set of constraints Σ. The construction of such procedures depends on the type of constraints allowed, and it will be discussed in Section 5.4.

Compute_Brute_Force orders the obtained sequences of defaults in descending lexicographic order. By *lexicographic order*, we mean the total order, denoted $<_s$, for the sequences of defaults in Δ induced by the order of the defaults in Δ. For example, we have $[\delta_1, \delta_2, \delta_4] <_s [\delta_1, \delta_3, \delta_4]$, since 124 < 134, and $[\delta_1, \delta_4] <_s [\delta_1, \delta_2, \delta_3]$, since 14 < 123.

PUC-Rio - Certificação Digital Nº 1112681/CA

```
void Compute_Brute_Force (T, Θ):
Input:    a default theory T=(V,Σ,Δ), where Δ is a finite set of generic defaults
Output:   a set Θ of maximal sequences of the defaults in Δ

1.  begin
2.    Construct the set L of all possible sequences of the defaults of Δ;
3.    Order the sequences in L by descending lexicographic order,
        creating a list L1,...,Lm;
4.    for each i=1 to m do                  /* Test consistency of Li against Σ */
5.      if Test-Consistency (Σ, cons(Li))
6.        then  Flag[i] = true;             /* Li is a candidate solution       */
7.        else  Flag[i] = false;            /* Li is not a candidate solution   */
8.
9.    Θ = ∅;
10.   for each i=1 to m do                  /* Test if each Li is maximal or not */
11.     if Flag[i]
12.       then begin
13.               Θ = Θ ∪ { Li };          /* Li is a maximal candidate solution */
14.               for each k = i+1 to m do  /* (ordered by descending lexic.)    */
15.                 if Lk is a subset of Li /* Lk is not a maximal candidate     */
16.                 then Flag[k] = false;   /* solution                          */
17.             end
18. end;
```

Figure 2. A brute-force algorithm to determine all maximal sequences of a set of defaults.

PUC-Rio - Certificação Digital Nº 1112681/CA

We now apply the brute force method to the example of Chapter 3.

Example 3: Suppose, again, for simplicity that the set of assertions $\boldsymbol{A}$ is:

$$\boldsymbol{A} = \{B(a), p(a,b), A(a), p(a,c)\}$$

Recall that the set of constraints is:

$\Sigma = \{ (\geq 1\ p^{-}) \sqsubseteq D,\ C \sqsubseteq (\geq 1\ p),\ C \sqsubseteq \neg(\geq 2\ p),\ A \sqsubseteq C,\ B \sqsubseteq C,\ A \sqsubseteq \neg B \}$

and that the set of defaults is $\Delta = \{\delta_1, \delta_2, \delta_3, \delta_4\}$, where:

δ_1 = “$: B(a) / B(a)$” δ_2 = “$: p(a,b) / p(a,b)$”

δ_3 = “$: A(a) / A(a)$” δ_4 = “$: p(a,c) / p(a,c)$”

Step 1: Construct the set L of all possible sequences of Δ. Order the sequences in L by descending lexicographic order in a list $L_1,...,L_m$;

In this step, as $n = 4$, the number C of sequences of Δ is $(4! / 3!) + (4! / 2!) + (4! / 1!) + (4! / 0!) + 1 = 65$.

In decreasing lexicographic order, the sequences are:

L = { $[\delta_4, \delta_3, \delta_2, \delta_1]$, $[\delta_4, \delta_3, \delta_1, \delta_2]$, $[\delta_4, \delta_2, \delta_3, \delta_1]$, $[\delta_4, \delta_2, \delta_1, \delta_3]$,
$[\delta_4, \delta_1, \delta_3, \delta_2]$, $[\delta_4, \delta_1, \delta_2, \delta_3]$, …, $[\delta_1, \delta_2, \delta_3, \delta_4]$,
$[\delta_1, \delta_3, \delta_2]$, $[\delta_1, \delta_2, \delta_4]$, $[\delta_1, \delta_2, \delta_3]$,
$[\delta_1, \delta_4, \delta_3]$, $[\delta_1, \delta_4, \delta_2]$, $[\delta_1, \delta_3, \delta_4]$,
$[\delta_2, \delta_3, \delta_1]$, $[\delta_2, \delta_1, \delta_4]$, $[\delta_2, \delta_1, \delta_3]$,
$[\delta_2, \delta_4, \delta_3]$, $[\delta_2, \delta_4, \delta_1]$, $[\delta_2, \delta_3, \delta_4]$,
$[\delta_3, \delta_2, \delta_1]$, $[\delta_3, \delta_1, \delta_4]$, $[\delta_3, \delta_1, \delta_2]$,
$[\delta_3, \delta_4, \delta_2]$, $[\delta_3, \delta_4, \delta_1]$, $[\delta_3, \delta_2, \delta_4]$,
$[\delta_4, \delta_2, \delta_1]$, $[\delta_4, \delta_1, \delta_3]$, $[\delta_4, \delta_1, \delta_2]$,
$[\delta_4, \delta_3, \delta_2]$, $[\delta_4, \delta_3, \delta_1]$, $[\delta_4, \delta_2, \delta_3]$,
$[\delta_1, \delta_4]$, $[\delta_1, \delta_3]$, $[\delta_1, \delta_2]$,
$[\delta_2, \delta_4]$, $[\delta_2, \delta_3]$, $[\delta_2, \delta_1]$,
$[\delta_3, \delta_4]$, $[\delta_3, \delta_2]$, $[\delta_3, \delta_1]$,
$[\delta_4, \delta_3]$, $[\delta_4, \delta_2]$, $[\delta_4, \delta_1]$,
$[\delta_4]$, $[\delta_3]$, $[\delta_2]$, $[\delta_1]$,
$\emptyset$}

Note that we have to consider the empty sequence, since it might be the case that no defaults can be fired.

Step 2: Test the consistency of *cons*(L_i) against the set of constraints Σ and mark each sequence with **true** or **false** (recall that each sequence represents a candidate solution).

In our example, the following sequences are marked with **true**:

$[\delta_1]$, $[\delta_2]$, $[\delta_3]$, $[\delta_4]$, $[\delta_1, \delta_2]$, $[\delta_1, \delta_4]$, $[\delta_2, \delta_3]$, $[\delta_2, \delta_4]$, $[\delta_3, \delta_4]$, …

and the following sequences are marked with **false**:

$[\delta_1, \delta_3]$, ...

Step 3: By descending lexicographic order, keep only in L the sequences of defaults that are the largest candidate solutions and add them to Θ. Finally, return to the user only the maximal candidate solutions:

Θ = { $[\delta_1, \delta_2]$, $[\delta_1, \delta_4]$,
$[\delta_2, \delta_3]$, $[\delta_2, \delta_4]$,
$[\delta_3, \delta_4]$ }

From Θ, we can build a set S of the maximal subsets of the consequents of simple defaults:

PUC-Rio - Certificação Digital Nº 1112681/CA

$S = \{\{B(a), p(a,b)\}, \{B(a), p(a,c)\},$
$\{p(a,b), A(a)\}, \{p(a,c), A(a)\},$
$\{p(a, b), p(a,c)\}\}$

5.2 Process Trees

In this section, we summarize a method to obtain extensions of (generic) default theories, called *process trees*, proposed by Antoniou (1999). Process trees offer a much more reasonable alternative to the brute force method, and yet compute all extensions of a (generic) default theory. Furthermore, the theory behind process trees does not assume that the set of defaults is finite, as in the brute force method described in Section 5.1. Again, Section 5.3 will discuss how to compute extensions of mashup default theories (with simple defaults only).

We recall that a (generic) *default* over an alphabet V is an expression of the form "$\varphi : \psi_1,...,\psi_n / \chi$" such that $\varphi, \psi_1,...,\psi_n$ and χ are closed predicate logic formulae over V and $n > 0$. The formula φ is the *prerequisite* (*pre* denotes this formula), $\psi_1...\psi_n$ are the *justifications* (*just* denotes this set of formulae), and χ is the *consequent* (*cons* denotes this formula) of the default.

Also recall that a default "$\varphi : \psi_1,...,\psi_n / \chi$" is *applicable* to a set of formulae Φ iff $\varphi \in \Phi$ and $\neg\psi_1 \notin \Phi,..., \neg\psi_n \notin \Phi$ (that is, each ψ_i is consistent with Φ).

In what follows, let $\boldsymbol{T} = (V, \Sigma, \Delta)$ be a default theory, where Δ is a (finite or infinite) set of (generic) defaults.

Let $\prod = (\delta_0, \delta_1, ...)$ be a finite or infinite sequence of (generic) defaults from Δ without multiple occurrences. Consider that $\prod$ is a possible order in which we fire some of the defaults in Δ. We associate two sets of first-order formulae with $\prod$, $In(\prod)$ and $Out(\prod)$, defined as follows:

PUC-Rio - Certificação Digital Nº 1112681/CA

- *In*(∏) = Th($\Sigma \cup$ {*cons*(δ) / δ occurs in ∏})

  In(∏) collects the information gained by the application of the defaults in ∏ and represents the current knowledge base after the defaults in ∏ have been fired.
- *Out*(∏) = {$\neg \psi$ / $\psi \in$ *just*(δ), for some δ occurring in ∏}

  Out(∏) collects formulae that should not turn out to be true, i.e., that should not become part of the current knowledge base, even after the defaults in ∏ have been subsequently fired.

We denote the initial segment of ∏ of length *k* by ∏[*k*], meaning that the length of ∏ is at least *k*.

We say that ∏ = $(\delta_0, \delta_1, \ldots, \delta_k, \ldots)$ is a *process* of ***T*** iff δ_k is applicable to *In*(∏[*k*]), for every $k \geq 0$.

Given a process ∏ of ***T***, we say that

- ∏ is *successful* iff *In*(∏) ∩ *Out*(∏) = ∅, otherwise it is *failed*.
- ∏ is *closed* iff every $\delta \in \Delta$ that is applicable to *In*(∏), δ already occurs in ∏.

The successful processes correspond to extensions, maximal or not. The closed processes capture the property of an extension being *closed under the application of defaults in Δ*; this means that we should not stop firing defaults until we are forced to (until we reach a contradiction), or until we fire all possible (applicable) defaults.

Finally, a set of formulae ***Γ*** is an *extension* of ***T*** iff there is some closed and successful process ∏ of ***T*** such that ***Γ*** = *In*(∏).

Based on the previous definitions, Antoniou (1999) proposed a canonical tree, called a *process tree*, which organizes all possible processes of ***T***. A process tree has the following characteristics:

- The nodes are labeled with two sets of first-order formulae (*In*-set and *Out*-set);
- The edges are generated by firing defaults and are labeled, at each step, with the name of the default that is being fired;
- The paths, starting at the root node, correspond to the processes of ***T***.

PUC-Rio - Certificação Digital Nº 1112681/CA

Furthermore, a process tree is correct and complete, as it computes all extensions of a default theory ***T***, according to Theorem 1.

Theorem 1: Let ***T*** = (V, Σ, Δ) be a default theory, where Δ is a (finite or infinite) set of (generic) defaults. If ∏ is a closed successful process of ***T***, then *In*(∏) is an extension of ***T***. Conversely, for every extension $\boldsymbol{\Gamma}$ of ***T***, there exists a closed, successful process ∏ of ***T*** with $\boldsymbol{\Gamma}$ = *In*(∏).

Proof:

(See Antoniou and Sperschneider (1993)). □

Example 4: Consider the default theory ***T*** = (V, Σ, Δ) with $\boldsymbol{\Sigma} = \varnothing$ and $\Delta = \{\delta_1, \delta_2\}$ with δ_1 = "*TRUE : p /¬q*" and δ_2 = "*TRUE : q / r*". The process tree (Figure 3) shows that ***T*** has only one extension.

For ∏ = (δ_1), we have *In*(∏) = $\{\neg q\}$ and *Out*(∏) = $\{\neg p\}$. As *In*(∏) ∩ *Out*(∏) = ∅, this process is successful. As there are no more defaults to apply, this process is closed.

For ∏ = (δ_2, δ_1), we have *In*(∏) = $\{r, \neg \boldsymbol{q}\}$ and *Out*(∏) = $\{\neg \boldsymbol{q}, \neg p\}$. As *In*(∏) ∩ *Out*(∏) = $\{\neg \boldsymbol{q}\}$, this process is failed.

So, the default theory ***T*** has only one extension, named *Th*($\{\neg q\}$).

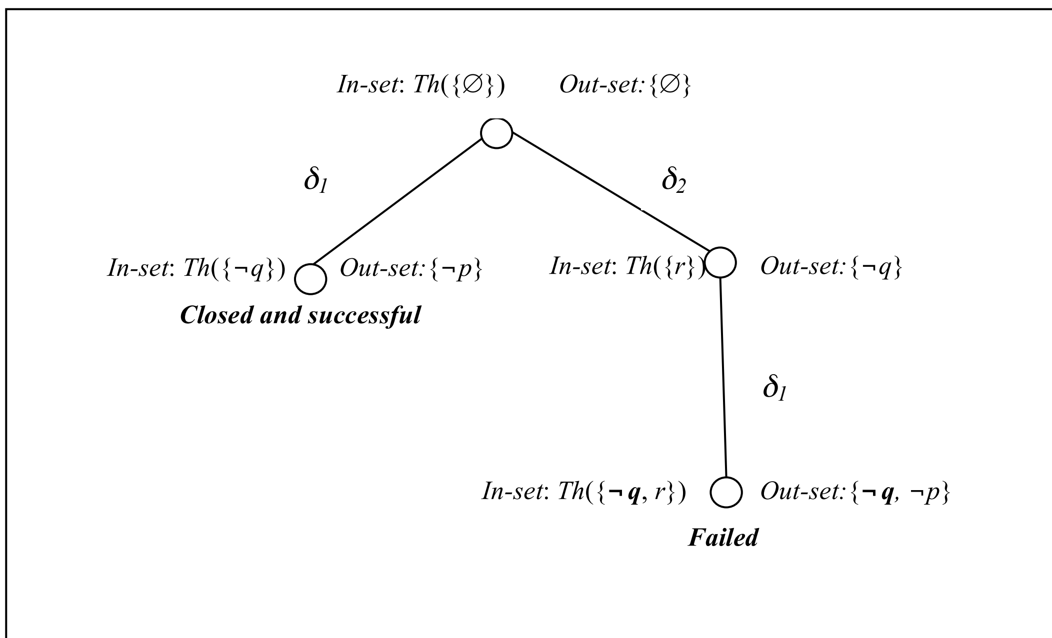

Figure 3. A process tree for Example 4.

PUC-Rio - Certificação Digital Nº 1112681/CA

5.3 Mashup Default Trees

In this section, we first show how to adapt the process trees introduced in Section 5.2 to mashup default theories. Inspired on such adaptation, we then propose a method to compute extensions of mashup default theories, called *mashup default trees*.

5.3.1 Process Trees for Mashup Default Theories

In what follows, let $\boldsymbol{T} = (V, \Sigma, \Delta)$ be a mashup default theory. Then, all defaults in Δ are simple, that is, of the form ":ψ / ψ".

The definitions in Section 5.2 then become much simpler. In particular, we can prove the following simple lemma.

PUC-Rio - Certificação Digital Nº 1112681/CA

Lemma 2: Let $\prod = (\delta_0, \delta_1, \ldots)$ be a finite or infinite sequence of defaults from Δ without multiple occurrences. Assume that Σ is consistent and that δ_k is of the form ":ψ_k / ψ_k", for every $k \geq 0$.

(i) $\prod$ is a process of $\boldsymbol{T}$ iff, for every k, $\neg \psi_k \notin In(\prod[k])$.

(ii) $\prod$ is a successful process iff $In(\prod)$ is consistent.

Proof:

Let $\prod = (\delta_0, \delta_1, \ldots)$ be a finite or infinite sequence of defaults from Δ without multiple occurrences. Assume that Σ is consistent and that δ_k is of the form ":ψ_k / ψ_k", for every $k \geq 0$.

(i) By definition of process, we have:

(1) $\prod = (\delta_0, \delta_1, \ldots, \delta_k, \ldots)$ is a process of $\boldsymbol{T}$ iff, for every $k \geq 0$, δ_k is applicable to $In(\prod[k])$

Now, for every $k \geq 0$, since δ_k is applicable to $In(\prod[k])$, we have:

(2) δ_k is applicable to $In(\prod[k])$ iff $\neg \psi_k \notin In(\prod[k])$

From (1) and (2), we have:

(3) $\prod$ is a process of $\boldsymbol{T}$ iff, for every k, $\neg \psi_k \notin In(\prod[k])$

(ii) First recall that:

(4) $In(\prod) = \text{Th}(\Sigma \cup \{\psi_k \,/\, \delta_k = \text{“}{:}\psi_k / \psi_k\text{”} \text{ occurs in } \prod\})$

(5) $Out(\prod) = \{\neg\psi_k \,/\, \text{for some } \delta_k = \text{“}{:}\psi_k / \psi_k\text{”} \text{ occurring in } \prod\}$

(⇒) Assume that ∏ is a successful process of ***T***. By definition of successful process, we have:

(6) $In(\prod) \cap Out(\prod) = \varnothing$

Hence, from (4), (5) and (6), we have:

(7) For every k, $\neg\psi_k \notin In(\prod)$

But, since Σ is consistent, by (7), we have:

(8) $In(\prod)$ is consistent.

(⇐) Assume that $In(\prod)$ is consistent. Then, we have:

(9) For every $\delta_k = \text{“}{:}\psi_k / \psi_k\text{”}$ that occurs in ∏, $\neg\psi_k \notin In(\prod)$

Hence, from (4), (5) and (9), we have:

(10) $In(\prod) \cap Out(\prod) = \varnothing$

Therefore, ∏ is a successful process of ***T***. □

Lemma 3: Let ∏ be a successful process of ***T***. Assume that Σ is consistent. Then, any other sequence ∏’ that has exactly the same defaults as ∏, perhaps in a different order, is also a successful process of ***T***.

Proof:

Let ∏ be a successful process of ***T*** and ∏’ be another sequence that has exactly the same defaults as ∏, perhaps in a different order. But, then $In(\prod) = In(\prod')$. By Lemma 2(ii), $In(\prod)$ is consistent, and so is $In(\prod')$. Again, by Lemma 2(ii), ∏’ is a successful process of ***T***. □

From Lemmas 2 and 3, we may immediately conclude that the process trees introduced in Section 5.2 can be simplified. We call *mashup default trees* the simplified version, which are constructed as follows:

PUC-Rio - Certificação Digital Nº 1112681/CA

- By Lemma 2(i), to construct a process, it suffices to test, for each node *N*, if $\neg\psi \notin$ Th($\Sigma \cup \{\psi$ / ":ψ / ψ" labels an edge of P_N }), where P_N is the path from the root to *N*.
- By Lemma 2(ii), to detect if a process is successful, if suffices to compute only the *In*-set of each node. Indeed, the *Out*-set is unnecessary, since it suffices to test the *In*-set for consistency.
- By Lemma 3, it suffices to consider the defaults in a pre-defined, arbitrary order. Then, it suffices to generate the edges by firing the simple defaults in the given order and label the edges with the name of the simple default that is being fired.

Again, the paths, starting at the root node, correspond to the processes of ***T***. Furthermore, for a fixed order of the defaults, the mashup default tree is unique. Formally, mashup default trees are defined as follows:

PUC-Rio - Certificação Digital Nº 1112681/CA

Definition 2: Let ***T*** = (V, Σ, Δ) be a mashup default theory. Assume that Σ is consistent and that $\Delta = \{\delta_1, ..., \delta_n\}$ is a finite, ordered set of simple defaults.

(i) The *mashup default tree* $\mathcal{T}_T$ for ***T*** is the n-ary tree, with an ordered set of children for each node and edges labeled with defaults from Δ, constructed as follows:

  (a) The root of $\mathcal{T}_T$ has *n* children and the edge from the root to the kth child is labeled with δ_k, for each $k \in [1,n]$.

  (b) Let *N* be an interior node of $\mathcal{T}_T$. Assume that the edge from the father of *N* to *N* is labeled with δ_j. Then, *N* has $(n - j)$ children, and the edge from *N* to the kth child is labeled with δ_k, for each $k \in [j+1,n]$.

(ii) Let *N* be a node of $\mathcal{T}_T$. Let Δ^N be the set of all defaults that label the edges of the path from the root of $\mathcal{T}_T$ to *N*. We say that Δ^N is the set of defaults *fired* up to *N*.

(iii) Let *N* be a node of $\mathcal{T}_T$. We say that *N* is a *successful node* iff $Th(\Sigma \cup \{\psi$ / ":ψ / ψ" occurs in Δ^N }) is consistent; otherwise we say that *N* is a *failure node*. □

The following theorem establishes some basic properties of mashup defaults trees that lead to the correctness of the procedure to compute extensions of mashup default theories presented in the next section.

Theorem 2: Let $\boldsymbol{T} = (V, \Sigma, \Delta)$ be a mashup default theory. Assume that Σ is consistent and that $\Delta = \{\delta_1, \ldots, \delta_n\}$ is a finite, ordered set of simple defaults. Let $\mathcal{T}_{\boldsymbol{T}}$ be the mashup default tree for $\boldsymbol{T}$. Then, we have:

(i) For any subset $\Delta' \subseteq \Delta$, there is a node N of $\mathcal{T}_{\boldsymbol{T}}$ such that $\Delta' = \Delta^N$.

(ii) If N is a successful node, then any ancestor of N is also a successful node.

(iii) If N is a failure node, then any descendent of N is also a failure node.

(iv) Γ is an extension of $\boldsymbol{T}$ iff there is a successful node N of $\mathcal{T}_{\boldsymbol{T}}$ such that $\Gamma = Th(\Sigma \cup \{\psi$ / ":ψ / ψ" occurs in $\Delta^N\})$ and there is no other successful node M of $\mathcal{T}_{\boldsymbol{T}}$ such that $\Delta^N \subset \Delta^M$ (that is, Δ^N is a strict subset of Δ^M).

PUC-Rio - Certificação Digital Nº 1112681/CA

Proof:

(i) Let $\Delta' \subseteq \Delta$. Assume that $\{\delta_{i_1}, \ldots, \delta_{i_k}\}$ is the set of all defaults in Δ', in the same order as in Δ. By Definition 2-(i-a), there is an edge labeled with δ_{i_1} from the root of $\mathcal{T}_{\boldsymbol{T}}$ to a node n_{i_1} of $\mathcal{T}_{\boldsymbol{T}}$ and, by Definition 2-(i-b), there is an edge labeled with $\delta_{i_{p+1}}$ from node n_{i_p} of $\mathcal{T}_{\boldsymbol{T}}$ to a node $n_{i_{p+1}}$ of $\mathcal{T}_{\boldsymbol{T}}$, for $p \in [1,k\text{-}1]$. So, there is a node $N = n_{i_k}$ of $\mathcal{T}_{\boldsymbol{T}}$ such that the edges in the path from the root of $\mathcal{T}_{\boldsymbol{T}}$ to N are labeled with $\delta_{i_1}, \ldots, \delta_{i_k}$. Therefore, we have that $\Delta^N = \{\delta_{i_1}, \ldots, \delta_{i_k}\}$, by Definition 2-(ii).

(ii) Follows directly from the definition of successful node. Indeed, if M is an ancestor of N, then $\Delta^M \subset \Delta^N$. Let $\Gamma^M = Th(\Sigma \cup \{\psi$ / ":ψ / ψ" occurs in $\Delta^M\})$ and $\Gamma^N = Th(\Sigma \cup \{\psi$ / ":ψ / ψ" occurs in $\Delta^N\})$. Then, since $\Delta^M \subset \Delta^N$, we have that $\Gamma^M \subset \Gamma^N$. Therefore, if Γ^N is consistent, so is Γ^M.

(iii) Follows likewise from the definition of failure node.

(iv) ($\Leftarrow$) Let N be a successful node N of $\mathcal{T}_{\boldsymbol{T}}$ and $\Gamma^N = Th(\Sigma \cup \{\psi$ / ":ψ / ψ" occurs in $\Delta^N\})$. Then, Γ^N is consistent, by definition of successful node. Assume that there is no other successful node M of $\mathcal{T}_{\boldsymbol{T}}$ such that $\Delta^N \subset \Delta^M$. Then, Δ^N is a

maximal set of defaults such that $Th(\Sigma \cup \{\psi$ / "$:\psi$ / ψ" occurs in $\Delta^N\})$ is consistent. Therefore, Γ^N is an extension of $\boldsymbol{T}$.

($\Rightarrow$) Let Γ be an extension of $\boldsymbol{T}$. Then, there is a maximal (finite) set Δ' of defaults in Δ such that $\Gamma = Th(\Sigma \cup \{\psi$ / "$:\psi$ / ψ" occurs in $\Delta'\})$ and Γ is consistent. By (i), there is a node N of $\mathcal{T}_T$ such that the path from the root of $\mathcal{T}_T$ to N is such that $\Delta^N = \Delta'$. Thus, $\Gamma = Th(\Sigma \cup \{\psi$ / "$:\psi$ / ψ" occurs in $\Delta^N\})$. Since Γ is consistent, N is a successful node. Furthermore, since $\Delta' = \Delta^N$ is maximal, there is no other successful node M of $\mathcal{T}_T$ such that $\Delta^N \subset \Delta^M$. □

5.3.2 Traversing a Mashup Default Tree

In this section, we discuss how to traverse a mashup default tree, using an intuitive example. But first we need to provide some additional concepts.

Backtracking is a general algorithm for finding solutions to some computational problems, notably constraint satisfaction problems, which incrementally builds candidates to the solution, and that abandons a candidate *c* ("backtracks") as soon as it determines that *c* cannot possibly be completed to reach a valid solution, because we have reached a contradiction.

A *depth-first search* (DFS) strategy consists in traversing a tree starting at the root node and exploring as far as possible the nodes along each branch, before moving back to the previous node, i.e., before *backtracking*.

We propose a method for traversing a mashup default tree in DFS. This method does not examine all possible scenarios, as the brute force method does, because it avoids building some branches of the tree (explained latter). Furthermore, it obtains all possible candidate solutions to the problem. However, it returns to the user only the maximal ones.

As in Section 5.1, we need to test the consistency of a set of assertions against a set of constraints. Such procedures will be discussed in Section 5.4.

Let $\boldsymbol{T}=(V,\Sigma, \Delta)$ be a mashup default theory and Δ be a finite set of simple defaults. This method obtains a set Θ of maximal subsets of the simple defaults in Δ.

At each step, the method fires the next simple default δ from Δ, considered a list, and calls a consistency test. If the test fails, the procedure backtracks to a

PUC-Rio - Certificação Digital Nº 1112681/CA

previous candidate solution. Otherwise, it continues firing other simple defaults from the list Δ, in a DFS manner, until there are no more simple defaults to fire. In this case, the procedure finds a subset of the simple defaults in Δ that represents a candidate solution (the set Θ of candidate solutions). Then, it goes back to the father node and starts again. Finally, it returns only the maximal candidate solutions (the maximal candidate solutions of Θ).

Before providing a concrete example, we note that a mashup default tree $\mathcal{T}$ has a typical topology, according to the number of considered simple defaults, and the order assumed for the defaults. For example, Figure 4 shows a mashup default tree generated from a set of four simple defaults, $\Delta = \{\delta_1, \delta_2, \delta_3, \delta_4\}$, considered in this order.

First observe that each default δ_i ($i = 1, \ldots, 4$) is fired once, creating 4 sub-trees of the tree $\mathcal{T}$ at the first level.

After firing default δ_i, only defaults δ_j such that $j > i$ are fired. For example, after we fire default δ_1 at the root node, we subsequently fire defaults δ_2, δ_3 and δ_4. After we fire default δ_2 at the root node, we can only fire defaults δ_3 and δ_4. However, after we fire default δ_3 at the root node, we can only fire default δ_4. Finally, after we fire default δ_4 at the root node, we have no more defaults to fire. Thus, the construction of the mashup default tree avoids building some branches.

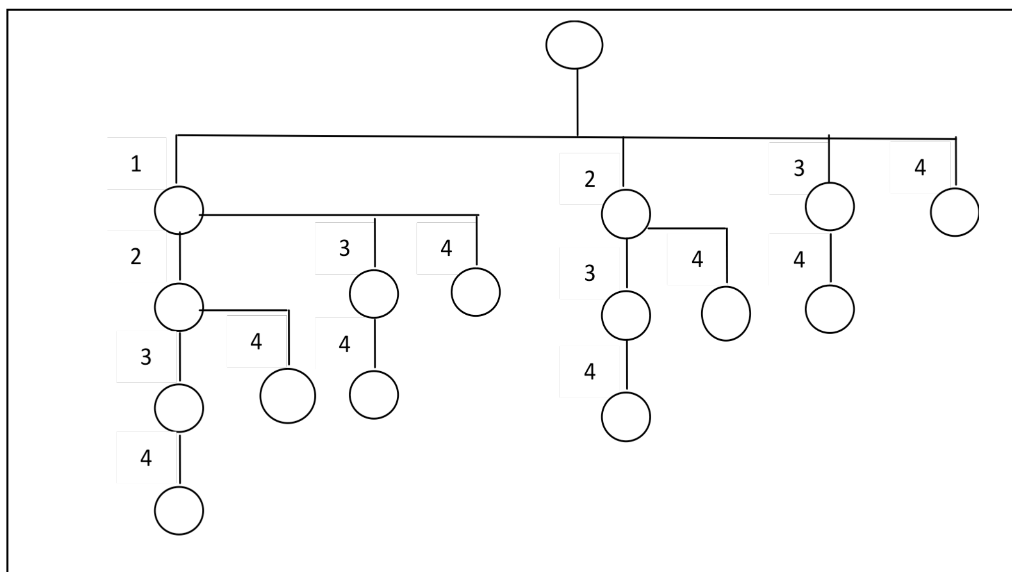

Figure 4. The typical topology of a mashup default tree with 4 defaults.

PUC-Rio - Certificação Digital Nº 1112681/CA

We now illustrate how this method executes with a concrete example.

Example 5: We refer again to the example of Chapter 3. Suppose, again, for simplicity that the set of assertions $\boldsymbol{A}$ is:

$$\boldsymbol{A} = \{B(a), p(a,b), A(a), p(a,c)\}$$

Recall that the set of constraints is:

$$\Sigma = \{ (\geq 1\ p^{-}) \sqsubseteq D,\ C \sqsubseteq (\geq 1\ p),\ C \sqsubseteq \neg(\geq 2\ p),\ A \sqsubseteq C,\ B \sqsubseteq C,\ A \sqsubseteq \neg B \}$$

Assume that the set of simple defaults is $\Delta = \{\delta_1, \delta_2, \delta_3, \delta_4\}$ where:

$\delta_1 =$ ": $B(a) / B(a)$" $\qquad \delta_2 =$ ": $p(a,b) / p(a,b)$"

$\delta_3 =$ ": $A(a) / A(a)$" $\qquad \delta_4 =$ ": $p(a,c) / p(a,c)$"

This method builds a mashup default tree from the set Δ. The prefixes of the branches of the tree correspond to the 2^n -1 = 15 subsets of Δ (see Figure 4). That is, the prefixes of the branches correspond to the set Θ of candidate solutions (which represent candidate extensions):

$$\Theta = \{ \{\delta_1\}, \{\delta_1, \delta_2\}, \{\delta_1, \delta_2, \delta_3\}, \{\delta_1, \delta_2, \delta_3, \delta_4\}, \{\delta_1, \delta_2, \delta_4\},$$
$$\{\delta_1, \delta_3\}, \{\delta_1, \delta_3, \delta_4\}, \{\delta_1, \delta_4\},$$
$$\{\delta_2\}, \{\delta_2, \delta_3\}, \{\delta_2, \delta_3, \delta_4\}, \{\delta_2, \delta_4\},$$
$$\{\delta_3\}, \{\delta_3, \delta_4\},$$
$$\{\delta_4\} \}$$

Then, we eliminate from Θ the subsets of defaults that are inconsistent (with respect to the set of constraints – see the example in Chapter 3), or that are not maximal. Finally, we return to the user only the maximal candidate solutions:

$$\Theta = \{ \{\delta_1, \delta_2\}, \{\delta_1, \delta_4\},$$
$$\{\delta_2, \delta_3\}, \{\delta_2, \delta_4\},$$
$$\{\delta_3, \delta_4\} \}$$

From Θ, we can build a set S of the maximal subsets of the consequents of simple defaults:

$$S = \{ \{B(a), p(a,b)\}, \{B(a), p(a,c)\},$$
$$\{p(a,b), A(a)\}, \{p(a,c), A(a)\},$$
$$\{p(a, b), p(a,c)\} \}$$

PUC-Rio - Certificação Digital Nº 1112681/CA

5.3.3 A Procedure to Compute Mashup Default Trees

In this section, we provide a description of the procedure for traversing a mashup default tree in DFS, called **Traverse_Mashup_Tree** (Figure 5). The major points of this version are discussed in what follows.

The set of solutions Θ is implemented using bit vectors of size *MAX*, the number of defaults in $\Delta=\{\delta_1,\delta_2,...,\delta_{MAX}\}$. A subset A of Δ is represented as a bit vector $A[0..MAX-1]$ such that $\delta_i \in A$ iff $A[i-1]$ = 1. The operation $subset(A, \Theta)$ tests if a set A is a subset of a set in Θ; the operation $add(A, \Theta)$ adds a set A to Θ; and the operation *cons(A)* transforms a set *A,* represented in a bit vector into a set C of consequents of the defaults in Δ such that $\alpha_i \in C$ iff $A[i-1]=1$ and δ_i = "$: \alpha_i / \alpha_i$" is the ith default in Δ.

The for loop on line 4 and the while loop on line 14 simulate a pre-order traversal of the mashup default tree, with the help of a stack *S* of integers. The operations *initialize(S)*, *n=top(S), n=pop(S)* and *push(n,S)* are as usual, while the operation *set(S)* transforms the integers in *S* into a set represented in a bit vector, respecting the order. For example, if the stack is $S=(1,3)$ and $MAX=4$, then the corresponding bit vector will be [1,0,1,0].

If it is consistent to fire all defaults from *M* to *MAX*, then $\Lambda = \{\delta_M, \delta_{M+1},..., \delta_{MAX}\}$ is a solution, if it is not a subset of a previous solution. Furthermore, firing all defaults from *N* to *MAX*, with $N>M$, will be a subset of Λ and therefore not a maximal solution. The process may then stop, as per lines 8 to 11.

Test-Consistency$(\Sigma, cons(set(S)))$ on line 20 tests if the consequents of the set of defaults represented in *S* are consistent with the set of assertions Σ. The consistency test is applied only if the set of defaults represented in *S* is not a subset of a solution, i.e., a set in Θ, as per line 18. If the consistency test succeeds, it indicates a solution only if the current set of defaults (represented in *S*) cannot be extended further, as per lines 23 and 31. Testing again, on line 25, if *set(S)* is not a subset of a solution is required because, even though this test has been performed in a previous step, the set of solutions might have been changed from the previous subset test to the new one (see Example 8).

Lastly, we observe that the correctness of the **Traverse_Mashup_Tree** procedure follows from the previous remarks and Theorem 2.

PUC-Rio - Certificação Digital Nº 1112681/CA

PUC-Rio - Certificação Digital Nº 1112681/CA

Traverse_Mashup_Tree (***T***, *Θ*):
Input: a mashup default theory $T=(V,\Sigma, \Delta)$, where $\Delta=\{\delta_1,\delta_2,...,\delta_{MAX}\}$ is a set of defaults
Output: a set *Θ* of maximal subsets of the consequents of the simple defaults in *Δ*

```
0.  begin
1.    Allocate a set of bit vectors Θ;               /* Θ represents the solutions found.               */
2.    Allocate a stack S of integers;                 /* S is a stack to help search for solutions.      */
3.    initialize(S);
4.    for M = 1 to MAX do                             /* MAX is the number of defaults in Δ.             */
5.    begin
6.        Allocate a bit vector A with all bits set,  /* A represents from the Mth default (bit M-1)     */
7.            from bit M-1 to bit MAX-1;              /*   to the last default (bit MAX-1).              */
8.        if ¬subset(A, Θ)                            /* If A is not a subset of a solution and          */
9.        then if Test-Consistency(Σ, cons(A))        /*   it is consistent to fire all such defaults,   */
10.             then begin add(A, Θ); exit end        /* then there are no more solutions to find.       */
11.       else exit;                                  /* (Likewise, when A is a subset of a solution).   */
12.       push(M,S);
13.           N = M;
14.       while ¬empty(S) do
15.           begin
16.               N = N + 1;
17.               push(N,S);
18.               if ¬subset(set(S), Θ)
19.               then    /* The stack S is not a subset of a solution (and hence must be tested for consistency)*/
20.                       if ¬Test-Consistency(Σ, cons(set(S)))
21.                       then begin /* The stack (with N on top) is not a consistent set of defaults        */
22.                               N = pop(S);           /* Remove N from the stack                          */
23.                               if N == MAX           /* If N is the last default,                        */
24.                               then begin            /* then the stack represents a solution,            */
25.                                   if ¬subset(set(S), Θ)  /* only if it is not a subset of a solution.   */
26.                                       then add(set(S), Θ);
27.                                           N = pop(S); /* Backtrack.                                     */
28.                                       end
29.                            end
30.                       else /* The stack is a consistent set of defaults (and is not a subset of a solution) */
31.                           if N == MAX               /* If N is the last default,                        */
32.                           then  begin add(set(S), Θ);  /* then the stack represents a solution.         */
33.                                       N = pop(S);   /* Backtrack twice, if possible.                    */
34.                                       if ¬empty(S) then N = pop(S);
35.                                 end
36.               else    /* The stack S is a subset of a solution (and hence consistent)                   */
37.                       if N == MAX                   /* If N is the last default,                        */
38.                       then begin N = pop(S);        /* then backtrack twice, if possible.               */
39.                                   if ¬empty(S) then N = pop(S);
40.                            end
41.           end
42.       end
43. end
```

Figure 5. An implementation of the depth-first algorithm to determine all maximal subsets of a set of defaults in *Δ*.

Example 6: The mashup default tree in Figure 7 and Table 8 indicate the result of running **Traverse_Mashup_Tree** for the mashup default theory ***T***$=(V,\Sigma,\Delta)$, where $\boldsymbol{\Sigma}$ = { $(\geq 1\ p^{-}) \sqsubseteq D$, $C \sqsubseteq (\geq 1\ p)$, $C \sqsubseteq \neg(\geq 2\ p)$, $A \sqsubseteq C$, $B \sqsubseteq C$, $A \sqsubseteq \neg B$ } and the set of assertions ***A*** = { *B(a), p(a,b), A(a), p(a,c)* }.

We recall that the constraint graph for the constraints is the one shown in Figure 6, and that the defaults are considered in the following order:

1) "*: B(a) / B(a)*"
2) "*: A(a) / A(a)*"
3) "*: p(a,b) / p(a,b)*"
4) "*: p(a,c) / p(a,c)*"

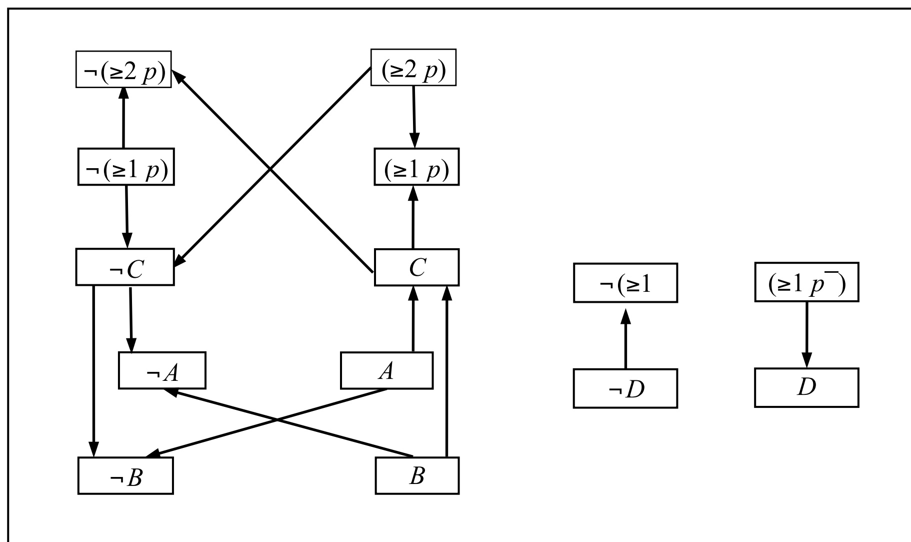

Figure 6. The constraint graph $G(\Sigma_M)$ for Σ_M.

Note that the topology of the mashup default tree in Figure 7 is the same as the one showed in Figure 4, as both trees have 4 defaults. The difference is that in Figure 7 some branches of the tree were not built (could be avoided).

PUC-Rio - Certificação Digital Nº 1112681/CA

The label of a node *N* in Figure 7 indicates the following:

- *S (Success)* – indicates that it is consistent to fire all defaults represented in the path from the root to *N*.
- *F (Fail)* – indicates otherwise.
- *C (Cut)* – indicates that *N* was not considered (*N* is a descendent of a node labeled with *F*).
- *A (Abandon)* – indicates that *N* was abandoned because, although it is consistent to fire all defaults represented in the path from the root to *N*, this is not a maximal solution, since it is a subset of another solution.

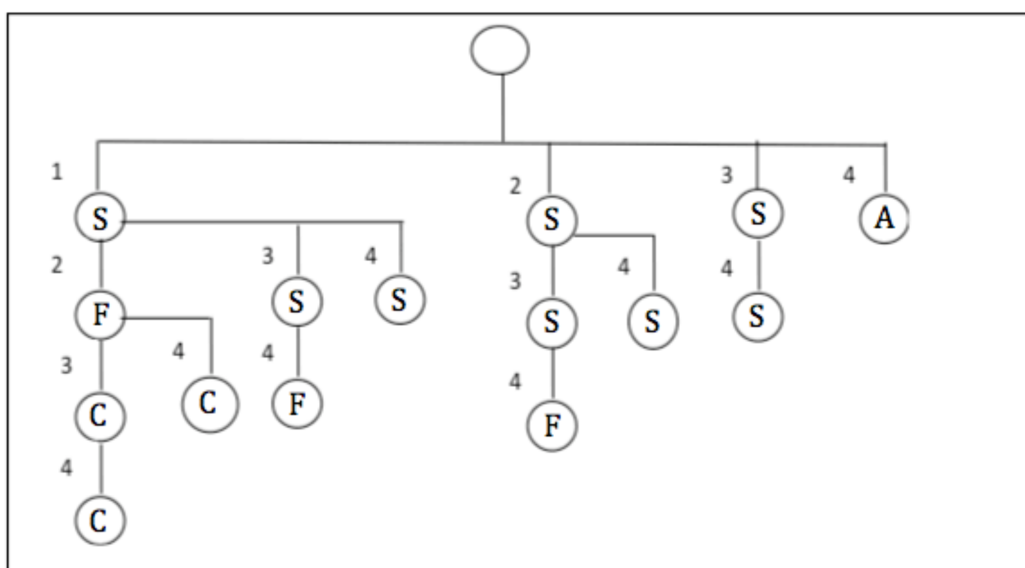

Figure 7. The mashup default tree for Example 6.

PUC-Rio - Certificação Digital Nº 1112681/CA

Table 8. A summary of the execution of Traverse_Mashup_Tree for Example 6.

M=1 (leftmost branch)				**M=2 (central left branch)**				**M=3 (central right branch)**			
Step	*Action*	*S*	*Test Result*	*Step*	*Action*	*S*	*Test Result*	*Step*	*Action*	*S*	*Test Result*
3	*init*	λ				λ				λ	
8	*subset?*		*fail*	8	*subset?*		*fail*	8	*subset?*		*fail*
9	*consistent?*		*fail*	9	*consistent?*		*fail*	9	*consistent?*		*success*
12	*push 1*	1		12	*push 2*	2		10	*add {3,4}*		
17	*push 2*	1 2		17	*push 3*	2 3			*exit*		
18	*subset?*	1 2	*fail*	18	*subset?*	2 3	*fail*	**M=4 (rightmost branch)**			
20	*consistent?*	1 2	*fail*	20	*consistent?*	2 3	*success*				
22	*pop -> 2*	1		37	*N=MAX?*		*fail*				
23	*N=MAX?*		*fail*	17	*push 4*	2 3 4					
17	*push 3*	1 3		18	*subset?*	2 3 4	*fail*				
18	*subset?*	1 3	*fail*	20	*consistent?*	2 3 4	*fail*				
20	*consistent?*	1 3	*success*	22	*pop -> 4*	2 3					
31	*N=MAX?*		*fail*	23	*N=MAX?*		*success*				
17	*push 4*	1 3 4		25	*subset?*	2 3	*fail*				
18	*subset?*	1 3 4	*fail*	26	*add {2,3}*	2 3					
20	*consistent?*	1 3 4	*fail*	27	*pop -> 3*	2					
22	*pop -> 4*	1 3		17	*push 4*	2 4					
23	*N=MAX?*		*success*	18	*subset?*	2 4	*fail*				
25	*subset?*	1 3	*fail*	20	*consistent?*	2 4	*success*				
26	*add {1,3}*	1 3		31	*N=MAX?*		*success*				
27	*pop -> 3*	1		32	*add {2,4}*	2 4					
17	*push 4*	1 4		33	*pop -> 4*	2					
18	*subset?*	1 4	*fail*	34	*pop -> 2*	λ					
20	*consistent?*	1 4	*success*								
31	*N=MAX?*		*success*								
32	*add {1,4}*	1 4									
33	*pop -> 4*	1									
34	*pop -> 1*	λ									

Notes:

a) The set {1,3} indicates that a maximal solution is obtained by firing the first and the third defaults, that is, a maximal set of assertions is *{B(a), p(a,b)}*. Furthermore, note that this set is represented as a bit vector [1,0,1,0], since there are 4 defaults.

b) The central right branch is not traversed, since *{3,4}* represents the solution Λ_{34} = {*p(a,b), p(a,c)*}, that is, since it is consistent to fire defaults δ_3 and δ_4. Therefore, the rightmost branch also does not need to be traversed, since it would generate a candidate solution, {*p(a,c)*}, generated by firing just the default δ_4, which is a subset of the solution Λ_{34}.

The complete set of solutions is:

{1,3} or {*B(a), p(a,b)*}

{1,4} or {*B(a), p(a,c)*}

{2,3} or {*A(a), p(a,b)*}

{2,4} or {*A(a), p(a,c)*}

{3,4} or {*p(a,b), p(a,c)*}

PUC-Rio - Certificação Digital Nº 1112681/CA

Example 7: The mashup default tree in Figure 8 and Table 9 indicate the result of running the **Traverse_Mashup_Tree** for the mashup default theory and the assertions of Example 7, but now assuming that the ordered set of defaults is:

1) "*: B(a) / B(a)*"
2) "*: p(a,b) / p(a,b)*"
3) "*: A(a) / A(a)*"
4) "*: p(a,c) / p(a,c)*"

This example illustrates that the order of firing the defaults leads to a different traversing of the mashup default tree, that is, to a different labeling of the nodes, but the solutions are the same.

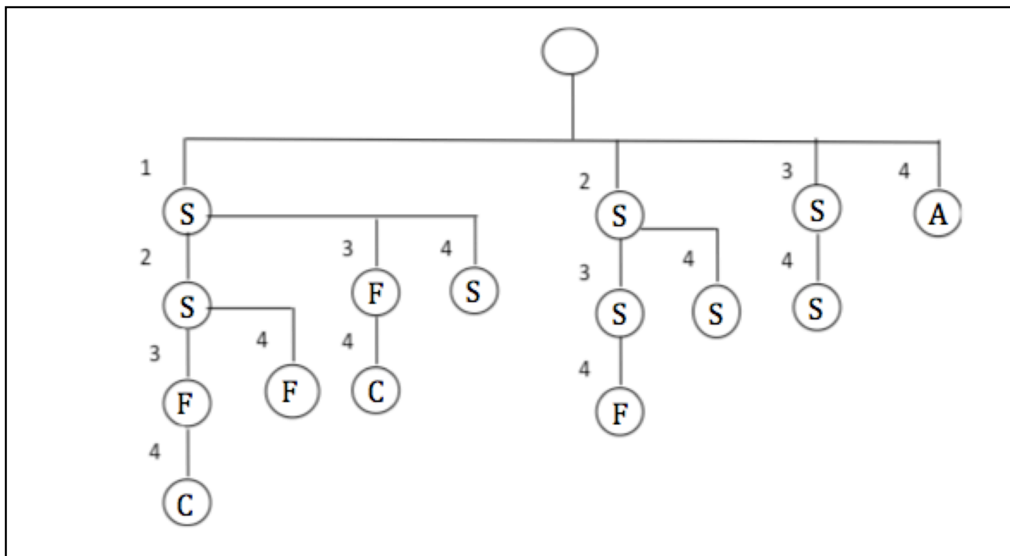

Figure 8. The mashup default tree for Example 7.

PUC-Rio - Certificação Digital Nº 1112681/CA

Table 9. A summary of the execution of Traverse_Mashup_Tree for Example 7.

***M*=1 (leftmost branch)**				***M*=2 (central left branch)**				***M*=3 (central right branch)**			
Step	*Action*	*S*	*Test Result*	*Step*	*Action*	*S*	*Test Result*	*Step*	*Action*	*S*	*Test Result*
3	*init*	λ				λ				λ	
8	*subset?*		*fail*	8	*subset?*		*fail*	8	*{3,4} subset?*		*fail*
9	*consistent?*		*fail*	9	*consistent?*		*fail*	9	*{3,4} consistent?*		*success*
12	*push 1*	1		12	*push 2*	2		10	*add {3,4}*		
17	*push 2*	1 2		17	*push 3*	2 3			*exit*		
18	*subset?*	1 2	*fail*	18	*subset?*	2 3	*fail*	***M*=4 (rightmost branch)**			
20	*consistent?*	1 2	*success*	20	*consistent?*	2 3	*success*				
31	*N=MAX?*		*fail*	31	*N=MAX?*		*fail*				
17	*push 3*	1 2 3		17	*push 4*	2 3 4					
18	*subset?*	1 2 3	*fail*	18	*subset?*	2 3 4	*fail*				
20	*consistent?*	1 2 3	*fail*	20	*consistent?*	2 3 4	*fail*				
22	*pop -> 3*	1 2		22	*pop -> 4*	2 3					
23	*N=MAX?*		*fail*	23	*N=MAX?*		*success*				
17	*push 4*	1 2 4		25	*subset?*	2 3	*fail*				
18	*subset?*	1 2 4	*fail*	26	*add {2,3}*	2 3					
20	*consistent?*	1 2 4	*fail*	27	*pop -> 3*	2					
22	*pop -> 4*	1 2		17	*push 4*	2 4					
23	*N=MAX?*		*success*	18	*subset?*	2 4	*fail*				
25	*subset?*	1 2	*fail*	20	*consistent?*	2 4	*success*				
26	*add {1,2}*	1 2		31	*N=MAX?*		*success*				
27	*pop -> 2*	1		32	*add {2,4}*	2 4					
17	*push 3*	1 3		33	*pop -> 4*	2					
18	*subset?*	1 3	*fail*	34	*pop -> 2*	λ					
20	*consistent?*	1 3	*fail*								
22	*pop -> 3*	1									
23	*N=MAX?*		*fail*								
17	*push 4*	1 4									
18	*subset?*	1 4	*fail*								
20	*consistent?*	1 4	*success*								
31	*N=MAX?*		*success*								
32	*add {1,4}*	1 4									
33	*pop -> 4*	1									
34	*pop -> 1*	λ									

Note: The central right branch is not traversed, since *{3,4}* represents the solution Λ_{34} = {*A(a), p(a,c)*}, that is, since it is consistent to fire defaults δ_3 and δ_4. Therefore, the rightmost branch also does not need to be traversed, since it would generate a candidate solution, {*p(a,c)*}, generated by firing just the default δ_4, which is a subset of the solution Λ_{34}.

The complete set of solutions is:

{1,2} or {*B(a), p(a,b)*}   {1,4} or {*B(a), p(a,c)*}

{2,3} or {*p(a,b), A(a)*}   {2,4} or {*p(a,b), p(a,c)*}

{3,4} or {*A(a), p(a,c)*}

that is exactly the same as in Example 6, as expected.

PUC-Rio - Certificação Digital Nº 1112681/CA

Example 8: The mashup default tree in Figure 9 and Table 10 indicate the result of running the **Traverse_Mashup_Tree** for the mashup default theory of Example 8, but assuming that the defaults (and their order) are:

1) “*: B(a) / B(a)*”
2) “*: p(a,b) / p(a,b)*”
3) “*: A(b) / A(b)*” (changed from *A(a)* to *A(b)*)
4) “*: p(a,c) / p(a,c)*”

This example illustrates that a slightly different default may lead to a rather different traversing of the mashup default tree.

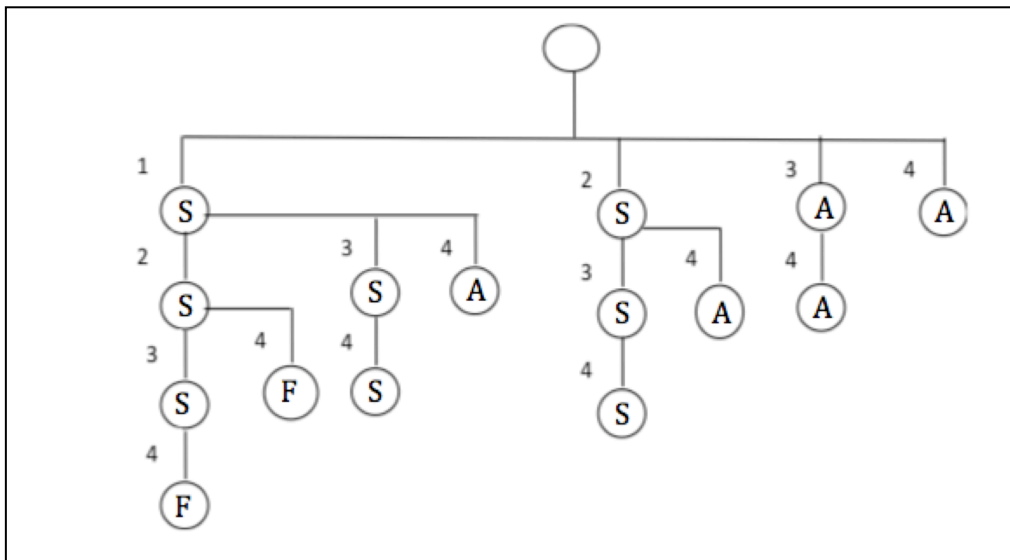

Figure 9. The mashup default tree for Example 8.

PUC-Rio - Certificação Digital Nº 1112681/CA

Table 10. A summary of the execution of Traverse_Mashup_Tree for Example 8.

***M*=1 (leftmost branch)**				***M*=2 (central left branch)**				***M*=3 (central right branch)**
Step	*Action*	*S*	*Test Result*	*Step*	*Action*	*S*	*Test Result*	***M*=4 (rightmost branch)**
3	*init*	λ				λ		
8	*subset?*		*fail*	8	*{2,3,4} subset?*		*fail*	
9	*consistent?*		*fail*	9	*{2,3,4} consistent?*		*success*	
12	*push 1*	1		10	*add {2,3,4}*			
17	*push 2*	1 2			*exit*			
18	*subset?*	1 2	***FAIL***					
20	*consistent?*	1 2	*success*					
31	*N=MAX?*		*fail*					
17	*push 3*	1 2 3						
18	*subset?*	1 2 3	*fail*					
20	*consistent?*	1 2 3	*success*					
37	*N=MAX?*		*fail*					
17	*push 4*	1 2 3 4						
18	*subset?*	1 2 3 4	*fail*					
20	*consistent?*	1 2 3 4	*fail*					
22	*pop -> 4*	1 2 3						
23	*N=MAX?*		*success*					
25	*subset?*		*fail*					
26	*add {1,2,3}*	1 2 3						
27	*pop -> 3*	1 2						
17	*push 4*	1 2 4						
18	*subset?*		*fail*					
20	*consistent?*		*fail*					
22	*pop -> 4*	1 2						
23	*N=MAX?*		*success*					
25	*subset?*	1 2	***SUCCESS***					
37	*N=MAX?*		*fail*					
27	*pop -> 2*	1						
17	*push 3*	1 3						
18	*subset?*		*fail*					
20	*consistent?*		*success*					
31	*N=MAX?*		*fail*					
17	*push 4*	1 3 4						
18	*subset?*		*fail*					
20	*consistent?*		*success*					
31	*N=MAX?*		*success*					
32	*add {1,3,4}*	1 3 4						
33	*pop -> 4*	1 3						
34	*pop -> 3*	1						
17	*push 4*	1 4						
18	*subset?*	1 4	*success*					
37	*N=MAX?*		*success*					
38	*pop -> 4*	1						
39	*pop -> 1*	λ						

PUC-Rio - Certificação Digital Nº 1112681/CA

Notes:

a) Note that the candidate solution represented by the set {1,2} is tested twice to verify if it is a subset of a solution (test results in boldface uppercase). Furthermore, note that the first test fails, whereas the second test succeeds. This illustrates why the subset test on line 25 is necessary.

b) The central left branch is not traversed since *{2,3,4}* represents the solution Λ_{234} = {*p(a,b), A(b), p(a,c)*}, that is, since it is consistent to fire defaults δ_2, δ_3 and δ_4. Therefore, the other branches also do not need to be traversed, since they would generate candidate solutions, which are subsets of the solution Λ_{234}.

The complete set of solutions is:

{1,2,3} or {*B(a), p(a,b), A(b)*}   {1,3,4} or {*B(a), A(b), p(a,c)*}

{2,3,4} or {*p(a,b), A(b), p(a,c)*}

5.3.4
Considerations on Heuristics for Building a Mashup Default Tree

Heuristics are designed to take advantage of partial knowledge that we have about the solution of the problem.

As we mentioned before, a mashup default tree can become prohibitively large, if we do not employed strategies to reduce it. Considering the restricted logics adopted in this text, one may suggest the following heuristics:

1) Order the data sources by priority, i.e., by importance or relevance. Then, give preference to data that comes from the higher priority data sources, when adding the data to the mashup.
2) Assume that the data that comes from each data source is consistent. Then, avoid redundant consistency tests for data that comes from the same data source.
3) Give preference to the simple defaults that, when fired, generate the largest number of derived assertions in the constraint graph.
4) Identify and isolate the simple defaults that cannot be fired together, as they generate assertions that are mutually inconsistent.

The first heuristics means that we should begin the construction of a mashup default tree by firing the simple defaults that belong to the most relevant data sources. So, we should begin by considering data coming from the first reliable data source and then progressively get data from the other sources, in order to enrich the data mashup. We regard that the priority of the data sources to be combined is provided by the user, according to his/her criteria like reliability and time of existence, and it is *static*, as it does not change with the time.

The second heuristics means that we can avoid making unnecessary consistency tests, if we assume that each data source returns data that is consistent with the constraints of the mashup ontology.

The third heuristics means that we should first fire the simple defaults that propagate the largest number of assertions in the constraint graph, that is, the

PUC-Rio - Certificação Digital Nº 1112681/CA

assertions that label nodes with the largest number of descendents in the constraint graph.

Finally, the forth heuristics means that we should analyze and separate the set of simple defaults that are mutually exclusive, as they would generate contradictory data. In fact, the construction of such tree branch can be avoided.

Following such heuristics, we can hopefully find solutions in a less expensive way.

5.4 Testing Consistency

5.4.1 Testing Consistency of OWL 2 Ontologies

In this section, we briefly summarize the results for the *Ontology Consistency Problem* for the OWL 2 profiles (OWL 2, 2012b).

In the context of mashup default theories, the *combined complexity*, that is, the complexity measured with respect to the total size of the axioms and the assertions in the ontology, is the most relevant, since we must test consistency of a set of constraints and assertions. Table 11 summarizes the combined complexity of the Ontology Consistency Problem, which has been widely covered in the literature for the various OWL 2 profiles (OWL 2, 2012b).

In general, if we assume that the mashup default theories use ontologies that satisfy a given profile, we may adopt the standard proof procedure for that profile to decide ontology consistency:

- OWL 2 EL: the tableaux procedure for the EL dialect (Baader et al., 2008), which is polynomial with respect to the size of the ontology.
- OWL 2 QL: the proof procedure sketched in (Calvanese et al., 2007), which is polynomial with respect to the size of the ontology.
- OWL 2 Direct Semantics: the proof procedure for the SHOIQ dialect (Horrocks and Sattler, 2005); even though the combined complexity is very high, this proof procedure has reasonable performance for typical OWL 2 ontologies (Horrocks and Sattler, 2005).

PUC-Rio - Certificação Digital Nº 1112681/CA

Table 11. Combined Complexity of the Ontology Consistency Problem for the OWL 2 Profiles.

Language	Combined Complexity
OWL 2 RDF-Based Semantics	Undecidable
OWL 2 Direct Semantics	N2EXPTIME-complete (NEXPTIME-complete if the property hierarchy can be translated into a polynomially-sized nondeterministic finite automaton)
OWL 2 EL	PTIME-complete
OWL 2 QL	NLogSpace-complete
OWL 2 RL	PTIME-complete
OWL 1 DL	NEXPTIME-complete

5.4.2 Testing Consistency of Lightweight Ontologies

In this section, we discuss how to test the consistency of $DL\text{-}Lite^{\mathcal{N}}_{core}$ ontologies, which we call *lightweight ontologies*, to avoid the somewhat awkward denotation $DL\text{-}Lite^{\mathcal{N}}_{core}$. We will work under the Unique Name Assumption (UNA). We show that constraint graphs lead to a consistent and complete decision procedure for the Lightweight Ontology Consistency Problem, which is cubic with respect to the total size of the axioms and the assertions in the ontology. The procedure is also amenable to be incrementally used in the context of mashup default trees.

Figure 10 shows the **Lightweight-Consistency** procedure that tests consistency of a set of positive assertions against a lightweight ontology. In this test, a set of assertions is used to populate a constraint graph. Theorem 3 establishes the correctness of the procedure and Theorem 4 its complexity.

Theorem 3: Let $\boldsymbol{O}=(V,\Sigma)$ be a lightweight ontology and $\boldsymbol{A}$ be a set of positive assertions over V. Assume that Σ is consistent. Under the Unique Name Assumption, procedure **Lightweight-Consistency**($\boldsymbol{O}$, $\boldsymbol{A}$) always stops and returns TRUE iff $\Sigma \cup \boldsymbol{A}$ has a model, and FALSE, otherwise.

Proof:

(i) Since the constraint graph for a lightweight ontology is acyclic, the procedure always stops.

PUC-Rio - Certificação Digital Nº 1112681/CA

(ii) Suppose that the procedure stops with FALSE. Then, there is a node *n* such that both *n* and its dual $\bar{n}$ are labeled with a constant "*a*". Hence, there is an expression *e* such than *e* labels *n* and $\bar{e}$ labels $\bar{n}$. But, we have that $u \sqsubseteq v$, $u(x) \models v(x)$. Hence, by definition of the procedure, specially rules CGR2 to CGR5, we may show that $\Sigma \cup \boldsymbol{A} \models e(a)$ and $\Sigma \cup \boldsymbol{A} \models \bar{e}(a)$. But this implies that $\Sigma \cup \boldsymbol{A}$ has no model.

(iii) Assume that Σ is consistent. Suppose that the procedure stops with TRUE. Then, there is no node *n* such that both *n* and its dual $\bar{n}$ are labeled with a constant "*a*". By modifying the construction of a Herbrand model for lightweight constraints (Casanova et al., 2010), under the Unique Name Assumption, it is possible to construct a model for $\Sigma \cup \boldsymbol{A}$. □

Theorem 4: Let $\boldsymbol{O}=(V,\Sigma)$ be a lightweight ontology and $\boldsymbol{A}$ be a set of positive assertions over *V*. Then, the **Lightweight-Consistency** procedure is $O(n(n^2+k))$, where *k* is the number of assertions in $\boldsymbol{A}$, *m* is the number of constraints in Σ and *n* is the number of distinct terms that occur in the constraints in Σ.

Proof:

Let *k* be the number of assertions in $\boldsymbol{A}$, *m* be the number of constraints in Σ and *n* be the number of distinct terms that occur in the constraints in Σ.

The construction of the constraint graph *G* in Step CGR0 takes time proportional to $(n(n+m))$ (Magalhães, 2015).

Using Warshall algorithm, the construction of G^* in Step CGR1 takes time proportional to n^3.

Steps CGR2 to CGR4 run in time proportional to $n.k$ since, for each of the *n* nodes, each for-loop can be performed by a simple scan over the *k* assertions.

Step CGR5 takes time proportional to n^2, which is the maximum number of edges of G*.

Finally, Step CGR6 takes time proportional to *n*, since we need to inspect all nodes of G*.

Thus, **Lightweight-Consistency** is $O(n(n^2+k))$, *since* $m \leq n^2$. □

PUC-Rio - Certificação Digital Nº 1112681/CA

Procedure Lightweight-Consistency(***O***, ***A***):
Input: a lightweight ontology $\boldsymbol{O}=(V,\Sigma)$ and a set of positive assertions ***A*** over *V*
Output: TRUE, if $\Sigma \cup \boldsymbol{A}$ has a model, and FALSE, otherwise

begin
CGR0. Construct the constraint graph $G=(N,E)$ for ***O***;
CGR1. Construct the transitive closure G^* of G;
for each node *N* of G^* **do**
begin
if *N* is labeled with a concept expression *u*
then for each assertion of the form *u(a)* in ***A*** **do**
CGR2. label *N* with the constant "*a*";
if *N* is labeled with a role expression of the form $(\geq n\ P)$
then for each set of assertions of the form $P(a,b_1),...,P(a,b_n)$ in ***A***, where $b_1,...,b_n$ are distinct constants **do**
CGR3. label *N* with the constant "*a*";
if *N* is labeled with a role expression of the form $(\geq n\ P^-)$
then for each set of assertions of the form $P(b_1,a),...,P(b_n,a)$ in ***A***, where $b_1,...,b_n$ are distinct constants **do**
CGR4. label *N* with the constant "*a*";
end
for each node *N* of G^* **do**
for each node *M* of G^* that is adjacent to *N* **do**
CGR5. Propagate all constants that label *N* to *M*;
for each node *N* of G^* **do**
CGR6. **if** *N* and its dual $\overline{N}$ are labeled with a constant "*a*",
then return *FALSE*;
CGR7. **return** *TRUE*;
end

Figure 10. A procedure to test the consistency of a set of positive assertions against a lightweight ontology.

The following examples illustrate how the procedure works and help understand the proof of Theorem 3. They also illustrate how the procedure can be used in conjunction with mashup default trees.

Example 9: Consider the following set Σ of constraints, whose constraint graph was shown again in this chapter (in Figure 6).

σ_1: $(\geq 1\ p^-) \sqsubseteq D$

σ_2: $C \sqsubseteq (\geq 1\ p)$

σ_3: $C \sqsubseteq \neg(\geq 2\ p)$

σ_4: $A \sqsubseteq C$

σ_5: $B \sqsubseteq C$

σ_6: $A \sqsubseteq \neg B$

and assume the following set Δ of defaults:

PUC-Rio - Certificação Digital Nº 1112681/CA

PUC-Rio - Certificação Digital Nº 1112681/CA

δ_1. *": B(a) / B(a)"*

δ_2. *": p(a,b) / p(a,b)"*

δ_3. *": p(a,c) / p(a,c)"*

Suppose that we fire all three defaults, generating the following set ***A*** of assertions:

1. *B(a)* (obtained by firing default δ_1)
2. *p(a,b)* (obtained by firing default δ_2)
3. *p(a,c)* (obtained by firing default δ_3)

The execution of Lightweight-Consistency for Σ and ***A*** generates the following steps (note that the procedure is non-deterministic, since the choice of which arc to process in CGR5 is non-deterministic):

4.	Label node *B* with "*a*"	1, by CGR2
5.	Label node ($\geq$1 *p*) with "*a*"	2, by CGR3
6.	Label node ($\geq$2 *p*) with "*a*"	2, 3, by CGR3
7.	Label node ($\geq$1 p^-) with "*b*"	2, by CGR4
8.	Label node ($\geq$1 p^-) with "*c*"	3, by CGR4
9.	Label node *C* with "*a*"	4, by CGR5
10.	Label node ¬*A* with "*a*"	4, by CGR5
11.	Label node ¬*C* with "*a*"	6, by CGR5
12.	Return FALSE	9, 11, by CGR6

Since Lightweight-Consistency returns FALSE, by Theorem 3, $\Sigma \cup$ ***A*** has no model. Then, it is inconsistent to fire defaults δ_1, δ_2 and δ_3 at the same time. □

Example 10: Consider again the same set Σ of constraints and the same set Δ of defaults as in Example 9. Suppose that we fire just the first two defaults, generating the set ***B*** of assertions:

1. *B(a)* (fire default δ_1)
2. *p(a,b)* (fire default δ_2)

The execution of Lightweight-Consistency for Σ and $\boldsymbol{B}$ generates the following steps:

3.	Label node B with "a"	1, by CGR2
4.	Label node ($\geq$1 p) with "a"	2, by CGR3
5.	Label node ($\geq$1 p^{-}) with "b"	2, by CGR4
6.	Label node C with "a"	3, by CGR5
7.	Label node $\neg A$ with "a"	3, by CGR5
8.	Label node D with "b"	5, by CGR5
9.	Label node $\neg$($\geq$2 p) with "a"	6, by CGR5
	(No new application of CGR5 is possible and the loop ends)	
10.	Return TRUE	by CGR7

Since Lightweight-Consistency returns TRUE, by Theorem 3, $\Sigma \cup \boldsymbol{B}$ has a model. Then, it is consistent to fire defaults δ_1 and δ_2 at the same time. Indeed, note that the above sequence of steps induces a model I for $\Sigma \cup \boldsymbol{B}$:

11.	$D^I = \{ \text{"}a\text{"}, \text{"}b\text{"} \}$	
12.	$a^I = \text{"}a\text{"}$ and $b^I = \text{"}b\text{"}$	
13.	$B^I = \{\text{"}a\text{"}\}$	by 1
14.	$P^I = \{(\text{"}a\text{"}, \text{"}b\text{"})\}$	by 2
15.	$C^I = \{\text{"}a\text{"}\}$	by 6
16.	$(\neg A)^I = \{\text{"}a\text{"}\}$	by 7
17.	$A^I = \{\text{"}b\text{"}\}$	by 11, 16
18.	$D^I = \{\text{"}b\text{"}\}$	by 8 □

The next example illustrates that the Lightweight-Consistency procedure stops even when the set of constraints has no finite model.

Example 11: Consider again the same set Σ of constraints and the same set Δ of defaults as in Example 9, with one additional constraint:

0. $D \sqsubseteq A$

Let Σ' be this new set of constraints. Then, the constraint graph for Σ' is the one in Figure 6, with an additional arc from the node labeled with D to the node labeled with A.

PUC-Rio - Certificação Digital Nº 1112681/CA

Suppose that we fire just the first two defaults, generating again the set ***B*** of assertions:

1. *B(a)* (fire default δ_1)
2. *p(a,b)* (fire default δ_2)

Then, the execution of Lightweight-Consistency for Σ and ***B*** generates the following steps:

3. Label node *B* with "*a*" 1, by CGR2
4. Label node (≥1 *p*) with "*a*" 2, by CGR3
5. Label node (≥1 p^{-}) with "*b*" 2, by CGR4
6. Label node *C* with "*a*" 3, by CGR5
7. Label node ¬*A* with "*a*" 3, by CGR5
8. Label node *D* with "*b*" 5, by CGR5
9. Label node ¬(≥2 *p*) with "*a*" 6, by CGR5
10. Label node *A* with "*b*" 8, by CGR5 (using the new arc)
11. Label node *C* with "*b*" 10, by CGR5
12. Label node ¬*B* with "*b*" 10, by CGR5
13. Label node (≥1 *p*) with "*b*" 11, by CGR5
14. Label node ¬(≥2 *p*) with "*b*" 11, by CGR5

(No new application of CGR5 is possible and the loop ends)

15. Return TRUE by CGR7

Since Lightweight-Consistency returns TRUE, by Theorem 3, $\Sigma' \cup$ ***B*** has a model. Hence, it is again consistent to fire defaults δ_1 and δ_2 at the same time, even in the presence of the new constraint "$D \sqsubseteq A$".

However, we note that $\Sigma' \cup$ ***B*** has no finite model, essentially because $\Sigma' \models (\geq 1\ p^{-}) \sqsubseteq (\geq 1\ p)$. Indeed, note that there is a path from the node labeled with (≥1 p^{-}) to the node labeled with (≥1 *p*) in the constraint graph for Σ' (which we recall is the constraint graph in Figure 6 with an additional arc from the node labeled with *D* to the node labeled with *A*). ☐

The next example provides a second situation that shows that finite and unrestricted logical implication differ for lightweight constraints, and yet that the **Lightweight-Consistency** procedure stops, even when the set of constraints has no finite model.

PUC-Rio - Certificação Digital Nº 1112681/CA

Example 12:

(a) Consider the following set of constraints Σ:

σ_1. $(\geq 1\ p) \sqsubseteq (\geq 1\ p^-)$

σ_2. $(\geq 2\ p) \sqsubseteq \bot$

σ_3. $(\geq 1\ p^-) \sqsubseteq (\geq 1\ p)$

Then, one may show that

1. $(\geq 1\ p) \sqsubseteq (\geq 1\ p^-)$ and $(\geq 2\ p) \sqsubseteq \bot$and $(\geq 1\ p^-) \sqsubseteq (\geq 1\ p)$

Indeed, let I be a finite interpretation of p. Recall that $(\geq 1\ p)^I$ is the domain of p^I and $(\geq 1\ p^-)^I$ is the range of p^I. Assume that I satisfies σ_1 and σ_2. Then, σ_2 forces p^I to be a function. In fact, we have that (where $|S|$ denotes the cardinality of a set S, as usual):

2. $|(\geq 1\ p)^I| \leq |(\geq 1\ p^-)^I|$ since I satisfies σ_1

3 $|(\geq 1\ p)^I| \geq |(\geq 1\ p^-)^I|$ since I satisfies σ_2 (forcing P^I to be a function)

4. $|(\geq 1\ p)^I| = |(\geq 1\ p^-)^I|$ by 2 and 3

5. $(\geq 1\ p)^I \subseteq (\geq 1\ p^-)^I \subseteq (\geq 1\ p)^I$ by 4, since I is finite

Hence, I satisfies $(\geq 1\ p^-) \sqsubseteq (\geq 1\ p)$ as desired.

Now, assume that K is such that $p^K = \{(2n,\ n)\ /\ n$ is a positive integer$\}$. Then, K satisfies σ_1 and σ_2, but not σ_3. Hence, we have:

6. $(\geq 1\ p) \sqsubseteq (\geq 1\ p^-)$ and $(\geq 2\ p) \sqsubseteq \bot$does not logically $(\geq 1\ p^-) \sqsubseteq (\geq 1\ p)$

(b) Assume that the set of constrains is $\boldsymbol{\Sigma} = \{\sigma_1, \sigma_2\}$ and that the set of assertions is $\boldsymbol{C} = \{p(a,b)\}$. Then, we have:

7. $p(a,b)$ assertion in $\boldsymbol{C}$

The execution of Lightweight-Consistency for $\boldsymbol{\Sigma}$ and $\boldsymbol{C}$ generates the following steps:

8. Label node $(\geq 1\ p)$ with "a" 7, by CGR3

9. Label node $(\geq 1\ p^-)$ with "b" 7, by CGR4

10. Label node $(\geq 1\ p^-)$ with "a" 8, by CGR5

(No new application of CGR5 is possible and the loop ends)

PUC-Rio - Certificação Digital Nº 1112681/CA

11. Return TRUE by CGR7

Thus, since Lightweight-Consistency stops and returns TRUE, $\Sigma \cup \boldsymbol{C}$ has a model (which is not finite, as proved above). □

5.4.3 Testing Consistency of Lightweight Ontologies in the Context of Mashup Default Trees

Given a mashup default theory $\boldsymbol{T} = (V, \Sigma, \Delta)$, the procedures introduced in Section 5.3 require testing the consistency of $\Sigma \cup cons(\Theta)$, where Θ is a subset of the set of defaults in Δ (and, we recall, $cons(\Theta)$ is the set of consequents of the defaults in Θ). Furthermore, the procedures require backtracking to a previous candidate solution, if the test fails.

Backtracking, in this case, can be efficiently implemented by modifying the **Lightweight-Consistency** procedure to incrementally consider a list $L_1,...,L_n$ of assertions and to backtrack from the processing of assertion L_j to the processing of assertion L_i, with $1 \leq i < j \leq n$. This can be efficiently implemented by sequentially tagging with integer k the constants that are propagated when processing assertion L_k, with $1 \leq k \leq n$, and by keeping the labeled graph from one call of the **Lightweight-Consistency** procedure to the next.

The following example illustrates the proposed change.

Example 13: Consider the same set Σ of constraints as in Example 6:

σ_1. $(\geq 1\ p^{-}) \sqsubseteq D$

σ_2. $C \sqsubseteq (\geq 1\ p)$

σ_3. $C \sqsubseteq \neg(\geq 2\ p)$

σ_4. $A \sqsubseteq C$

σ_5. $B \sqsubseteq C$

σ_6. $A \sqsubseteq \neg B$

Assume the following set Δ of defaults:

δ_1. $: B(a) \,/\, B(a)$

δ_2. $: p(a,b) \,/\, p(a,b)$

δ_3. $: p(a,c) \,/\, p(a,c)$

PUC-Rio - Certificação Digital Nº 1112681/CA

Suppose that we fire the defaults, one a time, generating the following list of assertions:

L1. *B(a)* (obtained by firing default δ_1)

L2. *p(a,b)* (obtained by firing default δ_2)

L3. *p(a,c)* (obtained by firing default δ_3)

Consider that we call Lightweight-Consistency for Σ and each of these assertions, in sequence. Also assume that we tag with *k* the constants propagated in the kth call:

First call of Lightweight-Consistency for Σ and L1:

4. Label node *B* with "a_1" L1, by CGR1
5. Label node *C* with "a_1" 4, by CGR4
6. Label node ¬*A* with "a_1" 4, by CGR4
7. Label node ¬(≥2 *p*) with "a_1" 5, by CGR4

(No new application of CGR4 is possible and the loop ends)

8. Return TRUE by CGR6

Second call of Lightweight-Consistency for Σ and L2 (reusing the graph left from the first call):

9. Label node (≥1 *p*) with "a_2" L2, by CGR2
10. Label node (≥1 p^{-}) with "b_2"L2, by CGR3
11. Label node *D* with "b_2" 10, by CGR4

(No new application of CGR4 is possible and the loop ends)

12. Return TRUE by CGR6

Third call of Lightweight-Consistency for Σ and L3 (reusing the graph left from the second call):

13. Label node (≥2 *p*) with "a_3" L2, L3, by CGR2
14. Label node ¬*C* with "a_3" 13, by CGR4
15. Return FALSE 5, 14, by CGR5

Since Lightweight-Consistency returns FALSE, we have to backtrack to the state after the second call. But this can be done by just dropping all constants with subscript "3" from the graph. □

PUC-Rio - Certificação Digital Nº 1112681/CA

5.5 Summary

Recall again from Section 3.8 the following question:

Q4. How to create a (maximal) consistent subset ***M*** of the set of the assertions collected from the data sources in such a way that ***M*** is consistent with the mashup constraints?

In Chapter 4, we discussed how to rephrase this question as the central problem of Default Logic, viz., how to compute extensions. In this chapter, we extensively discussed how to systematically search for extensions. Section 5.1 introduced a naïve, brute force method to compute extensions of a default theory with a finite set of generic defaults. Section 5.2 summarized a method to obtain extensions of (generic) default theories, called *process trees*, proposed by Antoniou (1999). Section 5.3 showed how to adapt process trees to mashup default theories. Inspired on such adaptation, we then proposed a new method to compute extensions of mashup default theories, called *mashup default trees* and extensively discussed procedures to explore such trees to compute extensions of mashup default theories.

The chapter concluded with a discussion on what we called the *Ontology Consistency Problem*, that is, the problem of testing the consistency of a set of assertions in the presence of a set of constraints, a central problem of the procedures introduced earlier in the chapter. Section 5.4.1 briefly summarized the results for the Ontology Consistency Problem for the OWL 2 profiles. Section 5.4.2 focused on the Lightweight Ontology Consistency Problem, that is, the Ontology Consistency Problem for $DL\text{-}Lite_{core}^{\mathcal{N}}$ ontologies, under the Unique Name Assumption. We showed that constraint graphs lead to a new consistent and complete decision procedure, which is polynomial with respect to the total size of the axioms and the assertions. The procedure is also amenable to be incrementally used in the context of mashup default trees.

PUC-Rio - Certificação Digital Nº 1112681/CA

6
Related Work

6.1
Consistent Mashups and Related Problems

In this thesis, we investigated the problem of building consistent data mashups from mutually inconsistent data sources.

Our approach first considers that the data sources are described by application ontologies, following a three-level ontology-based architecture. We assume that all ontologies are written in an expressive family of attributive languages and their sets of constraints are captured as constraint graphs. We reported a preliminary investigation on the generation of application ontologies in Sacramento et al. (2010), but without addressing the generation of their sets of constraints.

Our approach advocates that the data mashup service should have access to the constraints of the domain ontology to compute the constraints of the data mashup. Jain al. (2010) also argue that the Linked Open Data (LoD) Cloud, in its current form, is only of limited value for furthering the Semantic Web vision. They believe that it can be transformed from "merely more data" to "semantically linked data" by overcoming problems, such as schema heterogeneity and lack of schema level links.

In a previous work (Sacramento et al., 2012), we provided an ad hoc solution for the problem of constructing a consistent data mashup. We formalized the notion of data mashups in the context of knowledge bases, and provided a greedy algorithm based on an ordering of the data sources and on their corresponding assertions, induced by an ordering of the symbols in the alphabet (computed from the structure of the constraint graph). This thesis investigates the same problem using a new approach based on default theory, and provides an algorithm which employs some heuristics to optimize the problem.

Antoniou (Antoniou, 1999) proposed an operational model, called process

PUC-Rio - Certificação Digital Nº 1112681/CA

trees, which was used to obtain extensions of a given default theory. In our approach, we presented a model checker to compute extensions that was an adaptation of the process trees, called mashup default trees, which applies to a finite set of simple defaults. Furthermore, in our approach, the order of firing a set of simple defaults did not matter, as it lead to different ways of traversing a mashup default tree, but we continued to obtain the same solutions.

The problem of building consistent data mashups compares with the problems of filtering large schemas, querying inconsistent databases and data integration, among others. Below, we separately discuss each one of these problems.

Filtering large schemas. The data mashup service we propose allows browsing the domain ontology and choosing one or more concepts to be queried. From the concepts chosen, the service automatically queries the proper sources and combine the data returned in a way that only consistent data is exhibited to the user. So, it is based on the idea of *focus+context* (Villegas and Olivè, 2010), in which the notion of focus would be carried out by a vocabulary selection and the notion of context would be provided by the constraints. Note that this problem also appears in other information systems development activities, in which people need to operate with a fragment of the knowledge contained in the schema.

Querying inconsistent databases. Hogan (2011) and Lembo (2004) propose techniques to both repair databases and compute consistent query answers from the repaired database instances. The notion of data mashup we adopted uses a simple form of paraconsistent logical implication. De Amo et al. (2002) propose a tableau proof system for a paraconsistent logic to treat the integration process. However, their approach considers both positive and negative assertions, while our method only considers a positive set of assertions, as we work with defaults. Zhang et al. (2011) propose a tableau algorithm to implement paraconsistent reasoning in quasi-classical OWL. However, both methods are computationally hard.

Data integration. The problem of consistent data mashup is essentially equivalent to the problem of consistent data integration. As an example, Motro and Anokhin (2006) introduce fusion functions to treat attribute value's inconsistencies. The technique we propose considers other types of inconsistencies and could be modified to incorporate generalized fusion functions.

PUC-Rio - Certificação Digital Nº 1112681/CA

Wang et al. (2011) propose a solution that extends a data integration model with a data source quality criteria vector. Their strategy selects the "best" data source as the solution for the problem of data inconsistency, using a fuzzy multi-attribute decision making approach based on data source quality criteria. The model we propose, based on defaults, is far more general. It could be extended to consider priority defaults, where the priority is based on data quality and not on the relevance of the data sources.

With respect to data integration, the problem of constructing a data mashup service also compares with the usual problem of creating a data integration service, including a mediator, as follows.

Schema matching problem. We assume that each application ontology term matches the same term of the domain ontology, trivializing the problem of schema matching in our approach. Alternatively, the designer may use owl:sameAs statements to express that a term of the application ontology matches a term of the domain ontology. So, our data mashup service avoids the schema matching problem that traditional data integration services have been struggling with for some time (Bizer and Schultz, 2010).

Entity resolution problem. Our data mashup service may replace entity resolution by deductions involving owl:sameAs statements; for example, from the assertions *C(a)*, *P(a,b)* and *a* owl:sameAs *c*, the service may deduce *C(c)* and *P(c,b)*. By contrast, traditional data integration services typically include a (costly) entity resolution process to serve similar purposes. The works described in Jaffri et al. (2008) and Glaser et al. (2009) address the problem of identity resolution. These papers present some analysis of datasets used to link data and raise the question of how to manage the identity and meaning of URIs in the Semantic Web, in order to solve this problem.

Data Consistency problem. Our approach investigates the problem of constructing consistent data mashups when data to be combined is inconsistent with respect to a predefined set of constraints. It occurs because even if each data source returned data that is consistent with its own set of constraints, the combined data might be inconsistent.

PUC-Rio - Certificação Digital Nº 1112681/CA

6.2 Mashup Frameworks

For building data mashup applications, in many cases, there is a need for fetching and assembling pieces of information from multiple data sources. Mountantonakis et al. (2014) use the term *Semantic Warehouse* to refer to this concept.

Tran et al. (2014) and Hoang et al. (2014) overview the state-of-the-art in the area of Linked Data Mashups, describing technologies, applications and open challenges. They also compare mashups with traditional data integration tools, as mashups are technically similar, although philosophically quite different, and they attempt to move the control over data closer to the user and to his point of view.

Trinh et al. (Trinh et al., 2014) presents an approach for integrating multiple LOD datasets by levering their interconnections in a systematic and scalable manner, in order to support the flexible exploration of LOD by non-experts users, through the modularization of functionalities into blocks. However, in their approach, the developer has to manually guarantee that the output of his block adheres to the defined model, something that can cause inconsistencies.

Harth et al. (2013) shows an on-the-fly integration of static and dynamic sources for applications that consume Linked Data, which provides interactive access to data through integration pipelines that are executed at query time. They still need to investigate fault handling, in case of failing sources, and reduce the amount of network traffic, while keeping the query results update.

There are frameworks specifically designed to integrate Linked Data. The LOD2 project (Auer et al., 2012) includes the development of the *ODCleanStore framework* (Knap et al., 2012; Michelfeit and Knap, 2012), which offers linked data fusion at query time, resolving inconsistencies through pre-defined policies and generating provenance data to help users judge data quality and trustworthiness. This framework offers a data fusion and conflict resolution tool called *ODCS-FusionTool* (Michelfeit et al., 2014).

A similarly tool is *LDIF - Linked Data Integration Framework* (Schultz et al., 2011) that implements a mapping language to translate data from several sources to a mediated schema. It offers a service to identify when several distinct URIs refer to the same real-world object (entity resolution) and allows Linked Data quality assessment and data fusion through a tool named *Sieve* (Mendes et

PUC-Rio - Certificação Digital Nº 1112681/CA

al., 2012). We do not focus on the data fusion problem, but the technique we propose might be integrated into LDIF to include consistency checks.

Michelfeit and Mynarz (2014) present a more sophisticated approach for Linked Data fusion than the current available tools. They provide new data fusion features to address the problems of dependencies between RDF properties and the fusion of structured values. These extensions have been implemented in a tool named *LD-FusionTool*, which was developed as a part of the *UnifiedViews framework*, a successor of ODCleanStore.

Tzitzikas et al. (2014) describe the requirements, and present a process and a tool for constructing semantic warehouses. The authors developed a tool called *MatWare* (Materialized Warehouse) to validate the warehouse construction process. They also report their experience, using this tool, building a semantic warehouse for the marine domain.

Mynarz (2014) proposes a generic workflow for data integration, based on Linked Data and semantic web technologies, which covers schema alignment, data translation, entity reconciliation, and data fusion. He also presents an application that integrates public procurement data in order to improve data usability and value for analysis of such data.

Databugger (Kontokostas et al., 2014a) is a framework for test-driven quality assessment of Linked Data that ensures a basic level of quality by accompanying vocabularies, ontologies, and knowledge bases with a number of test cases (Kontokostas et al., 2014b). This methodology is centered on the definition of data quality integrity constraints and does not provide a complete reasoning and constraint checking, but allows verifying typical violations efficiently. As many datasets provide only limited schema information, they can perform automatic schema enrichment, suggesting schema axioms with a certain confidence value by analyzing the dataset (Kontokostas et al., 2014c). Our approach presupposes that the constraints of the data mashup are generated from the domain ontology constraints. From the obtained constraints, it allows checking the initial set of assertions coming from the data sources, obtaining new derived assertions, and finally analyzing and separating consistent assertions from inconsistent ones.

PUC-Rio - Certificação Digital Nº 1112681/CA

6.3 Working with Defaults

With respect to using defaults with a terminological knowledge representation formalism, Baader and Hollunder (1995) show that an extension of a decidable description logic becomes undecidable, if it is extended with (open) defaults, due to the unsatisfactory treatment of open defaults via Skolemization in Reiter's Semantics (Reiter, 1980). However, if the default only refers to known or named individuals, then decidability can be retained. So, under this restricted semantics, it is possible to compute all extensions of a finite terminological default theory.

Shaohua et al. (2008) present a framework for semantic query based on a prioritized default extension to description logic. They provide a mathematical foundation for querying information from incomplete description logic knowledge base by introducing prioritized default rules, in order to avoid conflict between them. Their approach gives preference to more specific defaults over more general rules, and considers both normal and not normal default rules. Our approach is neutral in this aspect, and the user may explicitly define the order of relevance of the data sources and it only works with normal (simple) defaults.

Sengupta et al. (2013) present an approach for integrating defaults with description logics that retain decidability, although working with free defaults. However, they imposed that exceptions to the default only occur in the named individuals of the knowledge base. Furthermore, their approach is *skeptical* as it requires a logical formula to be true in all the extensions of the default theory; while our approach is *credulous*, as it requires a set of assertion to be consistent in some extensions.

Finally, this thesis uses simple and closed defaults, as reasoning in this setting is decidable in general, when the underlying DL is also decidable. Furthermore, our defaults are grounded, as the extensions may only contain known individuals (or pairs of such, for roles). So, the problems faced by the extension of DL proposed by Baader and Hollunder (1995) are not applicable to our work.

PUC-Rio - Certificação Digital Nº 1112681/CA

7
Conclusions and Suggestions for Future Research

7.1
Conclusions

In this thesis, we focused on a new solution for the problem of building consistent data mashups from data sources that are mutually inconsistent with respect to a predefined set of constraints.

Remember that we raised the following questions in Chapter 3:

Q1. How to compute the mashup constraints from the domain ontology constraints?

Q2. How to match the data source vocabularies with the vocabulary of the domain ontology?

Q3. How to derive new assertions from those obtained from the data sources (after translation) and the mashup constraints?

Q4. How to create a (maximal) consistent subset ***M*** of the set of the assertions collected from the data sources in such a way that ***M*** is consistent with the mashup constraints?

Questions Q1 and Q2 were already discussed in previous work of the authors:

Question Q1 was presented in Casanova et al. (2011) and Sacramento et al. (2012) and Question Q2 was discussed in Sacramento et al. (2010). The matching step is in fact a trivial process, as we assumed that the vocabularies of the data sources are subsets of the vocabulary of the domain ontology.

We presented answers to Questions Q3 and Q4 in Chapters 4 and 5 of this thesis:

With respect to Question Q3, we showed in Chapter 4 how to translate an ontology and a set of assertions to a default theory. According to our definitions, a set of assertions retrieved from the data sources will be considered together iff the

PUC-Rio - Certificação Digital Nº 1112681/CA

corresponding simple defaults can be fired in the presence of the constraints. Furthermore, one should consider only maximal sets of such defaults to maximize the data retrieved from the data sources and shown to the user, without running into inconsistencies. We discussed, at the end of Chapter 4, how to rephrase the Question Q4 based on the use of Default Theories.

In Chapter 5, we introduced a brute force method to compute extensions of a default theory with a finite set of generic defaults. Next, we summarized a method to obtain extensions of (generic) default theories, called process trees (Antoniou,1999). We also showed how to adapt process trees to mashup default theories.

In this chapter, we also proposed a method to compute extensions of mashup default theories, called mashup default trees. Then, we provided a heuristic procedure to compute such mashup default trees. We showed that changing the order of firing the same set of (simple) defaults lead to a different traversing of a mashup default tree, that is, to a different labeling of its nodes, but the obtained solutions were the same. Finally, we discussed the problem of testing the consistency of a set of assertions in the presence of a set of constraints, called the Ontology Consistency Problem. In particular, we focused on this problem for $DL\text{-}Lite^{\mathcal{N}}_{core}$ ontologies, under the Unique Name Assumption, and showed that there is a consistent and complete decision procedure, and that the procedure is polynomial with respect to the total size of the axioms and the assertions.

The notion of consistent data mashup adopted uses a simple form of paraconsistent logical implication, as the incoming data was not removed, but tested and marked as “consistent” or “inconsistent”, before being combined.

We restricted our attention to simple defaults, as they were sufficient for formalizing the concept of a consistent data mashup. We observed that reasoning in this setting is decidable in general, when the underlying Default Logic is also decidable (Baader and Hollunder, 1995). We also verified that the simple default theory adopted always has extensions, i.e., although it is limited in expressiveness, it generates at least one extension, satisfying the semi-monotonicity property, as the normal default theories do (Antoniou, 1999).

Finally, we borrowed from Casanova et al. (2011) the technique to derive the constraints that must hold for a data mashup specification. The heuristic

PUC-Rio - Certificação Digital Nº 1112681/CA

procedure to compute consistent data mashups (extensions) is entirely novel, as it investigates the problem using a new approach based on default theory.

7.2 Suggestions for Future Research

As for future research, we suggest:

- The expansion of our results to consider equalities and inequalities assertions, which do not require the UNA (Unique Name Assumption) anymore;
- The expansion of our work to a more expressive class of ontologies, which include role hierarchies, using the results in Casanova et al. (2012);
- The treatment of the ontologies directly as default theories. In this case, mashups constraints would be considered as generic defaults;
- The use of the proposed heuristics for optimizing the procedure of computing consistent data mashups;
- The evaluation of the increased effectiveness of the procedure of computing consistent data mashups after applying the suggested heuristics;
- The presentation of the obtained extensions to the final user;
- The implementation of a prototype data mashup service that includes our approach based on simple defaults.

PUC-Rio - Certificação Digital Nº 1112681/CA

8
References

Antoniou, G. (1999) **A Tutorial on Default Logics**. ACM Computing Surveys, Vol. 31, No. 3, pp. 337-359.

Antoniou, G. and Sperschneider, V. (1993) **Computing Extensions of Nonmonotonic Logics**. In: Proceedings of the 4th Scandinavian Conference on Artificial Intelligence, IOS Press, pp. 20-29.

Arenas, M., Bertossi, L. and Chomicki, J. (1999) **Consistent Query Answers in Inconsistent Databases**. In: Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 68–79, ACM.

Artale, A., Calvanese, D., Kontchakov, R. and Zakharyaschev, M. (2009) **The DL-Lite Family and Relations**. J. of Artificial Intelligence Research. 36, 1–69.

Auer, S. et al. (2012) **Managing the Life-Cycle of Linked Data with the LOD2 Stack**. In: 2012 Int'l. Semantic Web Conference, pp. 1-16.

Baader, F., Brandt, S. and Lutz, C. (2008) **Pushing the EL Envelope Further.** In Proc. of the Washington DC workshop on OWL: Experiences and Directions (OWLED08DC).

Baader, F. and Hollunder, B. (1995) **Embedding Defaults into Terminological Knowledge Representation Formalisms**. Journal of Automated Reasoning 14(1), pp. 149–180.

Baader, F. and Nutt, W. (2003) **Basic Description Logics**. In: Baader, F.; Calvanese, D.; McGuiness, D.L.; Nardi, D.; Patel-Schneider, P.F. (Eds) The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge, UK.

Baader, F. and Sattler, U. (2000) **Tableau Algorithms for Description Logics**. In: R. Dyckhoff (ed.): Proc. of the Int. Conf. on Automated Reasoning with Tableaux and Related Methods (Tableaux 2000), Vol. 1847 of Lecture Notes in Artificial Intelligence, pp.1–18, Springer-Verlag.

PUC-Rio - Certificação Digital Nº 1112681/CA

Baier, C. and Katoen, J. (2008) *Principles of Model Checking,* MIT Press.

Bertossi, L. and Chomicki, J. (2003) **Query Answering in Inconsistent Databases**. Logics for Emerging Applications of Databases, pp. 43-83.

Bizer, C., Cyganiak, R. and Heath, T. (2007) **How to Publish Linked Data on the Web**. (Accessed Nov 12, 2014, http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/).

Bizer, C., Heath T., and Berners-Lee, T. (2009) **Linked Data - The Story So Far**. International Journal on Semantic Web and Information Systems (IJSWIS), Vol. 5, No. 3., pp. 1-22, doi:10.4018/jswis.2009081901

Bizer, C. and Schultz, A. (2010) **The R2R Framework: Publishing and Discovering Mappings on the Web**. First Int'l. Workshop on Consuming Linked Data (COLD 2010).

Bleiholder, J. and Naumann, F. (2005) **Declarative Data Fusion: Syntax, Semantics, and Implementation**, pp. 58-73.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M. and Rosati, R. (2007) **Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family**. Journal of Automated Reasoning 39(3):385-429. http://dx.doi.org/10.1007/s10817-007-9078-x.

Casanova, M.A., Breitman, K. K., Furtado, A. L., Vidal, V. M. P. and Macêdo, J.A.F. (2011) **The Role of Constraints in Linked Data**. In Proceedings of the 10th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2011).

Casanova, M.A., Breitman, K.K., Furtado, A.L., Vidal, V.M.P. and Macêdo, J.A.F. (2012) **An Efficient Proof Procedure for a Family of Lightweight Database Schemas**. In: M.G. Hinchey and L. Coyle (eds.), Conquering Complexity. London: Springer, (2012), 431–461.

Casanova, M.A., Lauschner, T., Leme, L. A. P. P., Breitman, K.K., Furtado, A. L. and Vidal, V.M.P. (2010) **Revising the Constraints of Lightweight Mediated Schemas**. Data & Knowledge Engineering, v.69, pp.1274 - 1301.

Chen, H., Lu, B., Ni, Y., Xie, G.T., Zhou, C., Mi, J. and Wu, Z. (2009) **Mashup by Surfing a Web of Data APIs**. PVLDB 2(2), pp.1602-1605.

PUC-Rio - Certificação Digital Nº 1112681/CA

De Amo, S. and Carnielli, W. (2002) **A Logical Framework for Integrating Inconsistent Information in Multiple Databases**. In: Int'l. Symposium on Foundations of Information and Knowledge Systems. LNCS, v. 2284. Berlin: Springer (2002), 67–84.

Emerson, E. A. (2008) **The Beginning of Model Checking: A Personal Perspective**, 2008. 25 Years of Model Checking, 5000: 27-45.

Fan, W., Geerts, F. and Xibei, J. (2008) **A Revival of Integrity Constraints for Data Cleaning**. Proceedings of VLDB´08, pp. 24-30.

Glaser, H., Jaffri, A. and Millard, I. (2009) **Managing Co-reference on the Semantic Web**. In: Proceedings of WWW2009 Workshop: Linked Data on the Web.

Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K.-U. and Umbrich, J. (2010) **Data Summaries for On-demand Queries over Linked Data**. In Proceedings of the 19th World Wide Web Conference (WWW2010), Raleigh, NC, USA, Apr.

Harth, A., Knoblock, C. A., Stadtmüller, S., Studer, R. and Szekely, P. (2013) **On-the-fly Integration of Static and Dynamic Sources**. In: 4th International Workshop on Consuming Linked Data. CEUR-WS.org.

Hartig, O. and Langegger, A. (2010) **A Database Perspective on Consuming Linked Data on the Web**. In Datenbankspektrum, 10(2), Oct. 2010.

Hoang, H.H., Cung, T.N.P., Truong, D. K., Hwang, D. and Jung, J. J. (2014) **Semantic Information Integration with Linked Data Mashups Approaches**. In: International Journal of Distributed Sensor Networks, vol. 2014, 12 pages.

Hogan, A. (2011) **Integrating Linked Data through RDFS and OWL: Some Lessons Learnt**. In Proceedings of the 5th International Conference on Web Reasoning and Rule Systems.

Holzmann, G.J. (2003) *The SPIN Model Checker: Primer and Reference Manual.* Addison-Wesley.

Horrocks, I., Kutz, O. and Sattler, U. (2006). **The Even More Irresistible SROIQ**. In Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006). AAAI Press.

PUC-Rio - Certificação Digital Nº 1112681/CA

Jaffri, A., Glaser, H. and Millard, I. (2008) **URI Disambiguation in the Context of Linked Data**, In Proceedings of the 1st Workshop on Linked Data on the Web.

Jain, P., Hitzler, P., Yeh, P.Z., Verma, K. and Sheth, A.P. (2010) **Linked Data is Merely More Data**. In: AAAI Spring Symposium 'Linked Data Meets AI' (2010), pp. 82–86.

Knap, T., Michelfeit, J., Daniel, J., Jerman, P., Rychnovsky, D., Soukup, T. and Necasky, M. (2012) **ODCleanStore: A Framework for Managing and Providing Integrated Linked Data on the Web**, In: Lecture Notes in Computer Science, Vol. 2012, Num. 7651, ISSN: 0302-9743, pp. 815-816.

Kontokostas, D., Brümmer, M., Hellmann, S., Lehmann, J. and Ioannidis, L. (2014) **NLP Data Cleansing Based on Linguistic Ontology Constraints**. In: Proceedings of the ESWC 2014, pp. 224-239.

Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J. and Cornelissen, R. (2014) **Databugger: A Test-Driven Framework for Debugging the Web of Data**. In: Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion, pp. 115-118. Republic and Canton of Geneva, Switzerland, International World Wide Web Conferences Steering Committee.

Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., Cornelissen, R. and Zaveri, A. (2014) **Test-Driven Evaluation of Linked Data Quality**. In: Proceedings of the 23rd International Conference on World Wide Web. Geneva: International World Wide Web Conferences Steering Committee, pp. 747-758.

Lembo, D. (2004) **Dealing with Inconsistency and Incompleteness in Data Integration**. Phd Thesis, Dipart. Informatica e Sistemistica, Università di Roma "La Sapienza".

Lenzerini, M. (2002) **Data Integration: A Theoretical Perspective**. Proceeding of ACM Symposium on Principles of Database Systems.

Levesque, H.J.; Brachman, R. J. (2004) *Knowledge Representation and Reasoning*. Amsterdam: Elsevier/Morgan Kaufmann.

PUC-Rio - Certificação Digital Nº 1112681/CA

Magalhães, R.C. (2015) *Operations over Lightweight Ontologies*. M.Sc. Dissertation, Department of Informatics, PUC-Rio, Rio de Janeiro, Brazil (Jan. 2015).

Mendes, P., Mühleisen, H. and Bizer, C. (2012) **Sieve: Linked Data Quality Assessment and Fusion**. Invited paper at the 1st International Workshop on Linked Web Data Management (LWDM 2012), Berlin, Germany, March 2012.

Merril, D. (2006) **Mashups: The New Breed of Web App – An introduction to Mashups**. (Acessed in March, 09th 2015, http://www.ibm.com/developerworks/web/library/x-mashups/index.html), 2006.

Michelfeit J. and Knap T. (2012) **Linked Data Fusion in ODCleanStore**. In: Proceedings of the ISWC 2012 Posters & Demonstrations Track, Boston, USA, CEUR Workshop Proceedings, ISSN: 1613-0073.

Michelfeit J., Knap T. and Necaský, M. (2014) **Linked Data Integration with Conflicts**. Journal of Web Semantics**.**

Michelfeit J. and Mynarz, J. (2014) **New Directions in Linked Data Fusion.** In: Proceedings of the ISWC 2014 Posters & Demonstrations Track, pp. 397-400.

Motro, A. and Anokhin, P. (2006) **Fusionplex: Resolution of Data Inconsistencies in the Integration of Heterogeneous Information Sources**. J. Information Fusion 7(2), pp. 176–196.

Mountantonakis, M., Allocca, C., Fafalios, P., Minadakis, N., Marketakis, Y., Lantzaki C. and Tzitzikas, Y. (2014) **Extending void For Expressing Connectivity Metrics of a Semantic Warehouse.** In Proceedings of the 1st International Workshop on Dataset PROFIling & fEderated Search for Linked Data co-located with the 11th Extended Semantic Web Conference, PROFILES@ESWC 2014.

Mynarz, J. (2014) **Integration of Public Procurement Data using Linked Data**. In: Journal of Systems Integration, Vol 5, No 4, pp. 19-31.

OWL 2 Web Ontology Language: Primer (Second Edition) Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, Sebastian Rudolph, eds. W3C Recommendation, 11 December 2012. Latest version available at http://www.w3.org/TR/owl2-primer/

PUC-Rio - Certificação Digital Nº 1112681/CA

OWL 2 Web Ontology Language: Profiles (Second Edition) Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, Carsten Lutz, eds. W3C Recommendation, 11 December 2012. Latest version available at http://www.w3.org/TR/owl2-profiles/

OWL 2 Web Ontology Language: Direct Semantics (Second Edition) Boris Motik, Peter F. Patel-Schneider, Bernardo Cuenca Grau, eds. W3C Recommendation, 11 December 2012. Latest version available at http://www.w3.org/TR/owl2-direct-semantics/

Reiter, R. (1980) **A Logic for Default Reasoning**. Artificial Intelligence 13(1-2), 81-132.

Sacramento, E.R., Casanova, M.A., Breitman, K.K., Furtado, A.L and Vidal, V.M.P. (2012) **Dealing with Inconsistencies in Linked Data Mashups**, In: 16th International Database Engineering Applications Symposium. New York, USA: ACM, 175–180.

Sacramento, E.R., Vidal, V. M. P., Macedo, J. A. F., Lóscio, B. F., Lopes, F. L. R. and Casanova, M. A. (2010) **Towards Automatic Generation of Application Ontologies**. Journal of Information and Data Management, v.1, pp.535 -550.

Schultz, A., Matteini, A., Isele, R., Bizer, C. and Becker, C. (2011) **LDIF - Linked Data Integration Framework**. In: 2nd Int'l. Workshop on Consuming Linked Data, Bonn.

Sengupta, K., Hitzler, P. and Janowicz, K. (2013) **Extending OWL with Defaults for Linked Data Alignmen**t.

Shaohua Liu, Junsheng Yu, Yinglong Ma, Bing Xu, Yuan Mai and Min Zhou (2008) **A Prioritized Default Extension to Description Logic Knowledge Base**. In: 5th International Conference on Fuzzy Systems and Knowledge Discovery.

Tran, T.N., Truong, D.K., Hoang, H. H. and Le, T. M. (2014) **Linked Data Mashups: A Review on Technologies, Applications and Challenges**. In: 6th Asian Conference on Intelligent Information and Database Systems, pp. 253-262. Springer, Heidelberg.

PUC-Rio - Certificação Digital Nº 1112681/CA

Trinh, T.D., Do, B.L., Wetz, P., Anjomshoaa, A., Kiesling, E. and Tjoa, A.M. (2014) **A Drag-and-block Approach for Linked Open Data Exploration**. In: 5th International Workshop on Consuming Linked Data (COLD'14).

Tzitzikas, Y., Minadakis, N., Marketakis, Y., Fafalios, P., Allocca, C., Mountantonakis, M. and Zidianaki, I. (2014) **MatWare: Constructing and Exploiting Domain Specific Warehouses by Aggregating Semantic Data**. In The Semantic Web: Trends and Challenges, volume 8465 of Lecture Notes in Computer Science, pages 721-736. Springer International Publishing.

Villadsen, J. (2002) **Paraconsistent Query Answering Systems**. In T. Andreasen, A. Motro, H. Christiansen, and H. L. Larsen, editors, In Proceedings of the 5th International Conference on Flexible Query Answering Systems, pp. 370–384.

Villegas, A. and Olivè, A. (2010) **A Method for Filtering Large Conceptual Schemas**. In: 29th Int'l. Conference on Conceptual Modeling (ER'10), Springer-Verlag, Berlin, Heidelberg, pp. 247-260.

Wang, X., Huang, L., Xu, X., Zhang, Y. and Chen, J. (2011) **A Solution for Data Inconsistency in Data Integration**. J. Information Science and Engineering 27(2), 681-695.

Yu, J., Benatallah, B., Casati, F. and Daniel, F. (2008) **Understanding Mashup Development**. IEEE Internet Computing 12(5): 44-52.

Zhang, X., Xiao, G. and Lin, Z. (2009) **A Tableau Algorithm for Handling Inconsistency in OWL**. In: 6th Annual European Semantic Web Conference, 399-413.

PUC-Rio - Certificação Digital Nº 1112681/CA