Linked service integration on the semantic web

Mahdi Bennara

N° d'ordre NNT : 2019LYSEI055

THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON
Opérée au sein de
L'INSA de Lyon

École Doctorale N° 512
Informatique et Mathématiques de Lyon

Spécialité/discipline de doctorat :
Informatique

Soutenue publiquement le 18/07/2019, par :
Mahdi BENNARA

Linked Service Integration on the Semantic Web

Devant le jury composé de :

MURISASCO, Elisabeth	Professeur	Univ. de Toulon	Rapporteure
FRONT, Agnès	MCF–HDR	Univ. de Grenoble	Rapporteure
MARET, Pierre	Professeur	Univ. de Saint-Étienne	Examinateur
SAVONNET, Marinette	MCF–HDR	Univ. de Bourgogne	Examinatrice
AMGHAR, Youssef	Professeur	Univ. de Lyon	Directeur de thèse
MRISSA, Michaël	Professeur	Univ. de Pau	Codirecteur de thèse

Laboratoire de recherche
Laboratoire d'InfoRmatique en Image et Systèmes d'information (LIRIS) - UMR 5205

Département FEDORA – INSA Lyon - Ecoles Doctorales – Quinquennal 2016-2020

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
CHIMIE	**CHIMIE DE LYON** **http://www.edchimie-lyon.fr** Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage **secretariat@edchimie-lyon.fr** INSA : R. GOURDON	M. Stéphane DANIELE Institut de recherches sur la catalyse et l'environnement de Lyon IRCELYON-UMR 5256 Équipe CDFA 2 Avenue Albert EINSTEIN 69 626 Villeurbanne CEDEX **directeur@edchimie-lyon.fr**
E.E.A.	**ÉLECTRONIQUE, ÉLECTROTECHNIQUE, AUTOMATIQUE** **http://edeea.ec-lyon.fr** Sec. : M.C. HAVGOUDOUKIAN **ecole-doctorale.eea@ec-lyon.fr**	M. Gérard SCORLETTI École Centrale de Lyon 36 Avenue Guy DE COLLONGUE 69 134 Écully Tél : 04.72.18.60.97 Fax 04.78.43.37.17 **gerard.scorletti@ec-lyon.fr**
E2M2	**ÉVOLUTION, ÉCOSYSTÈME, MICROBIOLOGIE, MODÉLISATION** **http://e2m2.universite-lyon.fr** Sec. : Sylvie ROBERJOT Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 INSA : H. CHARLES **secretariat.e2m2@univ-lyon1.fr**	M. Philippe NORMAND UMR 5557 Lab. d'Ecologie Microbienne Université Claude Bernard Lyon 1 Bâtiment Mendel 43, boulevard du 11 Novembre 1918 69 622 Villeurbanne CEDEX **philippe.normand@univ-lyon1.fr**
EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTÉ** **http://www.ediss-lyon.fr** Sec. : Sylvie ROBERJOT Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 INSA : M. LAGARDE **secretariat.ediss@univ-lyon1.fr**	Mme Emmanuelle CANET-SOULAS INSERM U1060, CarMeN lab, Univ. Lyon 1 Bâtiment IMBL 11 Avenue Jean CAPELLE INSA de Lyon 69 621 Villeurbanne Tél : 04.72.68.49.09 Fax : 04.72.68.49.16 **emmanuelle.canet@univ-lyon1.fr**
INFOMATHS	**INFORMATIQUE ET MATHÉMATIQUES** **http://edinfomaths.universite-lyon.fr** Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage Tél : 04.72.43.80.46 **infomaths@univ-lyon1.fr**	M. Luca ZAMBONI Bât. Braconnier 43 Boulevard du 11 novembre 1918 69 622 Villeurbanne CEDEX Tél : 04.26.23.45.52 **zamboni@maths.univ-lyon1.fr**
Matériaux	**MATÉRIAUX DE LYON** **http://ed34.universite-lyon.fr** Sec. : Stéphanie CAUVIN Tél : 04.72.43.71.70 Bât. Direction **ed.materiaux@insa-lyon.fr**	M. Jean-Yves BUFFIÈRE INSA de Lyon MATEIS - Bât. Saint-Exupéry 7 Avenue Jean CAPELLE 69 621 Villeurbanne CEDEX Tél : 04.72.43.71.70 Fax : 04.72.43.85.28 **jean-yves.buffiere@insa-lyon.fr**
MEGA	**MÉCANIQUE, ÉNERGÉTIQUE, GÉNIE CIVIL, ACOUSTIQUE** **http://edmega.universite-lyon.fr** Sec. : Stéphanie CAUVIN Tél : 04.72.43.71.70 Bât. Direction **mega@insa-lyon.fr**	M. Jocelyn BONJOUR INSA de Lyon Laboratoire CETHIL Bâtiment Sadi-Carnot 9, rue de la Physique 69 621 Villeurbanne CEDEX **jocelyn.bonjour@insa-lyon.fr**
ScSo	**ScSo*** **http://ed483.univ-lyon2.fr** Sec. : Véronique GUICHARD INSA : J.Y. TOUSSAINT Tél : 04.78.69.72.76 **veronique.cervantes@univ-lyon2.fr**	M. Christian MONTES Université Lyon 2 86 Rue Pasteur 69 365 Lyon CEDEX 07 **christian.montes@univ-lyon2.fr**

*ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

Acknowledgments

First of all, I would like to thank my respected supervisors Pr. Youssef AMGHAR and Pr. Michaël MRISSA for their support, confidence, patience and encouragement all along this long journey. Without their invaluable guidance and precious help this work would not have been achieved. I shall eternally be indebted to their teachings and assistance.

I would also like to thank the SOC research team and the LIRIS laboratory scientific staff for their valuable feedback, fruitful exchanges and for the very professional but also relaxed work environment, which was an incredible asset in helping achieve this work.

I convey my heartiest appreciation to Mahmoud, Pierre, Mehdi, Karim and Pierre-Antoine for their support, help and advice in writing and building many key components in this work. Special thanks to my dear friends Abdelmalek and Yaakoub for their assistance, for the shared good memories and for being there for me in the darkest of times.

Finally, I would like to express my deepest gratitude to Elisabeth MURISASCO and Agnès FRONT for accepting to review this work. I extend my heartfelt acknowledgement also to the examiners Pierre MARET and Marinette SAVONNET.

Mahdi BENNARA

To my parents, my wife and my daughter

Contents

List of Figures

List of Algorithms

List of Tables

Résumé

L'informatique orientée service facilite l'interopérabilité entre les systèmes distribués. Depuis quelques années, l'émergence du Web sémantique a posé de nouveaux défis pour la communauté de recherche dans les calculs et la compatibilité sémantique des données.

L'approche "services" et le Web sémantique constituent une piste prometteuse pour remédier aux problèmes qui entravent les deux domaines. D'une part l'orientation services permet d'assurer l'interopérabilité des données et des traitements au niveau sémantique, et d'autre part le Web sémantique permet d'automatiser les tâches de manipulation de services à un haut niveau.

Dans le cadre de notre travail de recherche, nous avons détaillé les défis que rencontre la communauté des chercheurs dans l'intégration des pratiques de l'orientation service dans le Web sémantique, et plus particulièrement l'intégration des services REST dans l'implémentation du Web qui repose sur les principes du "Linked Data" pour constituer ce que l'on appelle les "RESTful Linked Services". Les défis en question sont : La description, la découverte, la sélection et la composition.

Nous avons proposé une solution pour chacun de ces défis. Les contributions que nous avons proposées sont : la structure de descripteur, un algorithme de découverte sémantique, un algorithme de sélection basé sur Skyline et les répertoires de composition.

Nous pensons que l'ensemble de contributions que nous avons proposées peut être adopté par les fournisseurs de services sur le Web afin de faciliter l'intégration des pratiques du Web sémantique avec les technologies des services et de REST en particulier. Ceci permettra donc d'automatiser les tâches de manipulation de services a un haut niveau, tel que la découverte sur la base de concepts sémantiques, la sélection sur la base de propriétés non-fonctionnelles et de qualité de service et la composition de plusieurs services hétérogènes, sur le plan des données ainsi que sur le plan des traitements, afin d'obtenir des services composites avec de la valeur ajoutée.

Mots-clés: Services Web, Web sḿantique, Web des donnés

Abstract

Service Oriented Computing allows interoperability between distributed systems. In the last years, the emergence of the semantic Web opened new challenges for the research community regarding semantic interoperability on the data and processing levels.

The convergence of service orientation and the semantic Web together is a promising effort to solve the problems that hampered both research fields. On the one hand, service orientation allows interoperability on the data and processing levels, and on the other hand, semantic Web allows the automation of high-level service manipulation tasks.

In our research, we detail the challenges encountered by the research community to integrate the service orientation practices with the semantic Web, more precisely, integrating REST-based services with the semantic Web implementation based on Linked Data principles to obtain RESTful Linked Services. The challenges in question are : description, discovery, selection and composition.

We proposed a solution for each of these challenges. The contributions we proposed are : The descriptor structure, a semantically-enabled discovery algorithm, a Skyline-based selection algorithm and composition directories.

We think that these contributions can be adopted by service providers on the Web in order to allow a seamless integration of semantic Web practices with the service technologies and REST in particular. This allows the automation of high-level service manipulation tasks, such as semantically-enabled discovery, QoS-based selection and the composition of heterogeneous services, be it on the data or processing level, in order to create value-added composite services.

Keywords: Web services, Semantic Web, Linked Data

Chapter 1

Introduction

Over the past twenty years, service-oriented computing has promoted interoperability between distributed systems with the help of XML-based languages, protocols and tools (SOAP, WSDL, UDDI). Service-oriented architectures or SOAs rely on centralized approaches for basic service tasks such as discovery, selection and composition. The SOAs had success in the enterprise world, however they suffered from several problems such as the lack of semantic interoperability and scalability issues. These problems hampered their adoption on a large scale, and on the Web in particular, opening new challenges for the research community. At the same time, semantic Web has promoted the publication of data sets that refer to each other via interlinking, leading to a distributed dissemination of data accessible via RESTful APIs, also known as Linked Data [1].

Nowadays, the convergence of the semantic Web and Web services into a single framework enabling read/write access to Linked Data is an ongoing research challenge for the research community and normalization organizations [2]. Most proposed solutions rely on RDF-based services accessible through Web APIs. These services are called linked services as they are described with and exchange Linked Data, mostly through RESTful APIs. The mechanisms and algorithms for discovery, selection and composition typically adopted in SOAs need to be revisited and adapted to the distributed setup and large scale of the semantic Web that requires scalable solutions. Also, service descriptions must be adapted and enriched to make the semantics of services, data, operations and links explicit supporting the automation of discovery, selection and composition. In our research work, we propose solutions in the form of models and algorithms in order to address these open challenges and allow service technologies to integrate seamlessly with the practices of Linked Data and the semantic Web.

1. `http://linkeddata.org`
2. `https://www.w3.org/ns/ldp`

1.1 Global Context

The objective of this section is to present and define the general key concepts we use in our research. Section 1.1.1 presents the concept of Linked Data and the way service technologies handle Linked Data in order to create what we call Linked services. Section 1.1.3 presents the RESTful architectural style for building Web services and details how the different constraints contribute to the building of a Resource-oriented Web around Linked Data. Section 1.1.2 briefly presents the Semantic Web and the Web of Data and how they constitute the basis for building RESTful linked Web services. Section 1.1.4 introduces the concept of distributed affordance, an important key concept that changes the vision of building RESTful Web Services, whereby actions available for the user are automatically calculated and translated into operations that can be executed by the client that interacts with the Web services.

1.1.1 Linked Data and Linked Services

1.1.1.1 Linked Data

The term Linked Data [3] can be defined as a set of best practices for publishing and connecting structured data on the Web. These best practices have been adopted by an increasing number of data providers over the last years, leading to the creation of a global data space called the Web of Data. In summary, Linked Data is simply about using the Web to create typed links between data from different sources. These may be as diverse as data sources maintained by two organizations in different geographical locations, or simply heterogeneous systems within the same organization that have not easily inter-operated at the data level previously.

Technically, Linked Data refers to data published on the Web in such a way that it is machine-readable, its meaning is explicitly defined, it is linked to other external data sources and can in turn be linked to from external data sources. While the primary units of the hypertext Web are HTML [4] documents connected by untyped hyperlinks, Linked Data relies on documents containing data in RDF [5] format. However, rather than simply connecting these documents, Linked Data uses RDF to make typed statements that link arbitrary things in the world. The result, which is referred to as the Web of Data, may more accurately be described as a web of things in the world, described by data on the Web [Bizer et al., 2009b].

3. `https://www.w3.org/DesignIssues/LinkedData.html`
4. HyperText Markup Language : `https://www.w3.org/html`
5. Resource Description Framework : `https://www.w3.org/RDF/`

The principles upon which Linked Data structures data, as outlined by its inventor Tim Berners-Lee, are the following :

— Use URIs [6] as names for things.
— Use HTTP URIs so that people can look up those names.
— When someone looks up a URI, provide useful information using the standards (RDF and SPARQL [7]).
— Include links to other URIs, so that they can discover more things.

Linked Data uses a well-established stack of technologies :

— URI for identifying data in the form of resources.
— RDF for representing and interlinking data in the form of triples or assertions annotated semantically using ontologies and vocabularies.
— SPARQL for querying RDF graphs and extracting data from identified resources.
— HTTP [8] for data retrieval, exchange and manipulation using the basic CRUD (Create, Read, Update, Delete) operations.

Navigation through Linked Data in the Web of Data is very similar to navigation through pages in the classic Web of hypertext documents. The difference is that there are more challenges to overcome on the Web of Data due to the structuring and linking of data in RDF compared to HTML documents. The Web of Data exposes machine-readable data and allows flexible and automatic data processing and reasoning [Wilde, 2010a] [Domingue et al., 2011].

1.1.1.2 Linked Services

A service, in general, can be defined as a software component in a distributed system that provides the possibility of executing a business activity, where its functionality as well as its inputs and outputs are well defined. Linked services can be seen as the combination of service technologies and Linked Data practices. The increasing popularity of services on the Web has exemplified the need for standardized mechanisms for publishing structured data ; Linked Data was a great success in that regard. On top of that, RESTful Web services (cf. section 1.1.3) rely on a stack of technologies very similar to Linked Data, namely URIs for identification and HTTP for resource manipulation. This natural synergy gave birth to Linked Services as a powerful tool for designing distributed applications in an environment like the Web of Data.

The vision towards Linked Services is based on two simple ideas : publishing service

6. Uniform Resource Identifier : `https://tools.ietf.org/html/rfc3986`
7. Simple Protocol And RDF Query Language : `https://www.w3.org/TR/sparql11-query/`
8. HyperText Transfer Protocol : `https://tools.ietf.org/html/rfc7231`

descriptions in the Web of Data and creating services for the Web of Data (i.e., services that process Linked Data and generate Linked Data). In a nutshell, Linked Services are services described with Linked Data that consume and produce Linked Data. Therefore, these service descriptions whereby their inputs and outputs, their functionality, their non-functional properties and their links are described in terms of RDF-based vocabularies and exposed following Linked Data principles, and thus contribute to expanding the information space of the Web of Data. Up until recent years, the Web of Data has been constrained to be a read-only information space where applications retrieve and display data without altering it. The advent of service technologies and their adoption on the Web scale has enabled read/write access to the wealth of information in the Web of data.

As such, Linked Service descriptions represent highly valuable information which is still to be provided in the Web of Data : data about reusable functionality on the Web. In addition to that, thanks to these descriptions, Linked Services are therefore services that can consume RDF from the Web of Data and can also generate additional RDF to be fed back to the Web of Data. In other words, the advent of Linked Data has sparked the need for a processing layer on top of the wealth of information currently available in the Web of Data, which remains relatively unexploited [Pedrinaci et al., 2010a] [Domingue et al., 2011].

1.1.2 Semantic Web and the Web of Data

The seeds of the idea which became known as the semantic Web can be traced back to the earliest days of the World Wide Web by its inventor Tim Berners-Lee :

> *Evolution of objects from being principally human-readable documents to contain more machine-oriented semantic information, allowing more sophisticated processing* [Berners-Lee et al., 1994].

> *The first step is putting data on the Web in a form that machines can naturally understand, or converting it to that form. This creates what I call a Semantic Web - a web of data that can be processed directly or indirectly by machines* [Berners-Lee and Fischetti, 2000].

Semantic Web is an evolution of the classic Web of documents where data is semantically annotated with machine-readable and processable information allowing automation of software functionality. Semantically annotated data offers a lot of potential in building Web applications. These annotations allow many interactions to be automated thus excluding the need for human intervention. For example, annotated links guide Web crawlers and annotated sets of data help Web applications determine how to automatically

process them. These semantics can be utilized in conjunction with advanced reasoning mechanisms and similarity interlinking to connect resources to each other opening new possibilities to access and process them.

The Web of services has started an evolution towards semantic-level interoperability, with a lot of work around semantically described Web services [Martin et al., 2004] [Roman et al., 2005] [Kopeckỳ et al., 2007] [Vitvar et al., 2008] to allow services to exchange semantically annotated data. Combining the REST architectural style with semantic annotations unlocks the full benefits of using Linked Data for Web applications. Automating discovery and composition of RESTful services with the help of semantic Web technologies is a key challenge to exploit the full potential of the semantic Web. In fact, the research community is interested in making the same transition the Web of documents made from the static read-only Web 1.0 to the dynamic read/write Web 2.0 (where users are central), on the currently static and read-only Web of Data. The use of RESTful services is regarded as the final ingredient to make this transition and allow dynamic access to the wealth of Linked Data. Conversely, RESTful services are more focused on describing resources than on linking the data within them and most current REST APIs are described using natural language (documentations) and could really benefit from Linked Data.

Recent advances in the semantic Web research area have been promoting Linked Data [Bizer et al., 2009a] and a set of technologies, languages and tools such as JSON-LD [Sporny et al., 2014], RDF [Lassila and Swick, 1999], OWL [9] [Schreiber and Dean, 2004], SPARQL [Prud'hommeaux and Seaborne, 2008] and POWDER [10] [Archer et al., 2009] that allow the annotation of data, resources and services on the Web with explicit and machine-readable semantics. This evolution towards Linked Data gave birth to the Web of Data which is considered the most successful approach in establishing a semantic Web as imagined by [Berners-Lee and Fischetti, 2000]. The Web of Data, or Web of Linked Data, shares the following properties with the classic Web of documents :

— It is generic and can contain any type of data.
— Anyone can publish data on it.
— Data publishers are not constrained by which data type or vocabulary to represent and annotate their data with.
— Entities are connected by RDF links, creating a global graph that spans existing data sources and enables discovery of new ones.

From an application development point of view the Web of Data has the following characteristics :

— Data is strictly separated from formatting and presentation aspects.

9. Web Ontology Language
10. Protocol for Web Description Resources

— Data is self-describing, meaning if an application encounters data described with an unfamiliar vocabulary, it should easily be able to dereference the URIs that identify the terms in order to find their definitions.
— The use of HTTP as a standardized data access mechanism and RDF as a standardized data model simplifies data access.
— The Web of data is open meaning applications do not have to be implemented to only work with a fixed set of data sources, but can discover new data sources at runtime by following RDF links.

1.1.3 RESTful Linked Web Services

Contrary to classic RPC [11] oriented services, a RESTful Web service is a Web service bound by the constraints of the REST [12] architectural style [Fielding, 2000] and utilizing the stack of Web technologies for identification of resources (URI), data exchange (HTTP) and data serialization (XML, JSON, HTML, etc.) . A RESTful linked Web service is further bound by the constraints and principles of Linked Data (cf. section 1.1.1).

During the last few years, both the overall number of Web APIs exposed on the Web [13] and the increasing proportion of RESTful APIs has shown the interest of a resource-oriented Web [Bülthoff and Maleshkova, 2014]. The success of RESTful Web services is highlighted via Web sites such as *www.programmableweb.com* that referenced 103 APIs available on the Web by the end of 2005 and more than 14903 RESTful APIs in 2017, representing roughly 81,53% of all registered APIs [14].

REST imposes a set of constraints to designing and building APIs, the most relevant of these, in the context of our research, are [Wilde, 2010b] :

— Resource Identification : Anything that is available for interactions should have an identifier.
— Uniform Interface : Interaction with identified resources should be based on a uniform interface so that anything that is identified is readily available for interaction.
— Self-describing messages : interactions should be based on exchanges of messages (documents or data) which should be labelled with their type, and therefore can be treated by the corresponding software (typically a parser).
— Hypermedia driven application state : messages should be based on data formats

11. Remote Procedure Call
12. REpresentational State Transfer
13. `http://www.programmableweb.com/api-research`
14. `https://www.programmableweb.com/news/which-api-types-and-architectural-styles-are-most-used/research/2017/11/26`

that may contain typed links, and interactions with RESTful services means following those links according to the goal of the service consumer and the semantics of the link types.

— Stateless interaction : interactions are independent from each other. This decouples interactions and enables loose coupling between clients and servers. Servers do not need to remember clients, simplifying their design, and clients can take advantage of scalability optimizations such as caching and replication, enhancing performance.

The adherence to the REST constraints guarantees desirable architectural properties, the most relevant to our research are :

— Scalability : Scalability can be defined as the ability of a system to handle an increasing load of computation without a loss in efficiency, while keeping the potential for future extensions. The REST architectural style brings loose coupling to the Web environment via client/server architecture and stateless interactions which simplifies the design of flexible applications and high-performance services while encouraging reuse.

— Semantic interoperability on the Web : Semantic interoperability opens up a lot of opportunities allowing the exchange of machine-readable information and enabling high-level computations such as inferencing, knowledge extraction and computable logic. SOAP-based Web services have helped reaching syntactic level interoperability for distributed applications on the Web. The REST architectural style revisited the way we interact with services, highlighting important constraints such as uniform interface (and consequently generic clients), hypermedia-driven applications (HATEOAS [15]) and cacheability. With the help of Linked Data practices (as we mentioned in section 1.1.1), RESTful Web services show promising potential for enabling semantic interoperability in the Web of Data, which we intend to exploit in our research work.

In the remainder of this thesis (unless explicitly specified), the terms *(Web) service*, *(Web) resource* and *(Web) API* are used interchangeably and refer to RESTful linked Web Services. In fact, when a server offers such a resource-centric and hypermedia-based API, the concept of service as an invokable functionality gradually starts to fade, since the client sees nothing but resources and the relationships between them and only interacts with them through said API [Verborgh et al., 2012].

15. Hypermedia As The Engine Of the Application State

1.1.4 Distributed Affordance

In the context of the Web of resources, an affordance is the perceived action opportunity provided by a resource. The idea behind the concept of distributed affordance [Verborgh et al., 2013] is to dynamically create usage opportunities (affordances) for resources based on the information already present in their representation combined with knowledge from distributed sources including the user's preferences and independent, cross-application action providers. This augments the affordances given by the representation with additional controls that directly relate to the representation itself (such as a specific book title), instead of merely to its context (books in general).

In more technical words, distributed affordance consists in automatically extending the HATEOAS notion of REST by making actions from distributed providers on the Web, other than those proposed by the application state (the resource publishers), available in an actionable form such as a hyperlink or a form. In this case, the only information needed from the resource publisher is the representation of the resource, there is no need for the publishers to know, a priori, what are the actions the user desires to perform.

The main objective of the distributed affordance concept is to help solve discrepancy problems between affordance publishers and consumers. Based on the user's profile and preferences, the most relevant affordances are constructed using the information about his current browsing context. For example, for the user reading a book review, hyperlinks to the e-book version and the user's local library could be inserted automatically as distributed affordance to enable the user to easily and directly buy the book from the local library or view an abstract online, if possible.

The technical challenges to construct distributed affordances are :

1. Extracting the non-actionable information from the representation of the resource.
2. Organizing the knowledge about actions offered by providers.
3. Capturing the user's preferences and context.
4. Combining the non-actionable information and the knowledge about action providers into possible actions.
5. Integrating affordances for these actions into the original resource representation to enable easy and direct use.

Distributed affordance is a key concept that allows us to view from a different perspective the Web of data and Web of services in order to fully exploit their potential. The relevance of the distributed affordance concept to our research lies in the fact that it enables to dynamically discover usage opportunities for certain services which can be potentially useful in a composition scenario where a specific functionality is required.

Additionally distributed affordance takes advantage of the semantic annotations of representations of resources and services and their discovery (cf. section 1.3.1 and 1.3.2), which makes it a good use case to measure the applicability of solutions we propose to address the important challenges in our research.

1.2 Motivating Scenario

The objective of this section is to identify the key problems tackled in this thesis, which are related to discovering and composing RESTful Web services, with the help of a book selling scenario. This scenario illustrates and motivates the contributions of our research. The scenario involves a human user, a Web client and the Web. The user wants to buy a set of books and have them shipped to his address. He expresses his need via a request written to the Web client. The Web client processes the request and extracts information that helps it find the services to answer the user's needs. The Web client also stores data about the user preferences in terms of QoS [16], which are taken into account when processing the request.

1.2.1 Scenario Organization

In order to understand how the scenario works, we need to detail each actor and his role in this series of interactions.

The disposal of the scenario setup is detailed as follows :

— The user accesses the Web through his device that could be any type of terminal able to access the Web (desktop computer, laptop, smartphone, tablet, etc.).
— The machine client is the software program responsible for interacting with the user from one side thanks to a GUI [17], and the Web on the other using the HTTP protocol. It is hosted on the user's device. Most commonly, this is either a Web browser or a mobile application with support for add-ons.
— The set of services involved in the book selling scenario are accessed by the client through the Web. These services are designed and maintained by their respective publishers, which are in most cases day-to-day businesses.

The Web services needed for this scenario are grouped in three different sets : competing book selling services, competing shipping services and competing online payment

16. Quality of Service
17. Graphical User Interface

services [18]. Every Web service has its own quality of service properties. The different Web services, which are instantiated by a set of Web resources [19], are detailed in the following :

— The competing book selling Web services offer the same functionality and services, but differ in QoS aspects and business-related data (book inventory for example). They allow the user to browse books in their store, select few, place an order, edit their shopping cart, etc.
— The competing shipping Web services allow users to specify an address and select a delivery option for their online orders to be shipped together with the details of their order and vendor (in this case the chosen book selling service).
— The competing online payment Web services are responsible for the secure money transfer from the user's bank account (through credit card, or paypal for example) to the different vendors and shipping service providers, they also manage other related activities such as billing, refunds, etc.

The setup and interactions of the scenario is shown in Fig 1.1. The user interacts with the client using its GUI. The client in turn interacts with the services on the Web starting with the book selling service initially (the entry point). The services provide links to other services (these are RDF links, formally speaking) that help the user achieve his goal, in this instance : the book selling service cannot function without a shipping service since the books in this scenario are physical entities and have to be delivered in a materialized way. Also, both the book selling and the shipping services need a payment service in order to ensure the transfer of money from the user's bank account to their own for revenue generation. However, it is very important to note that these links are not hard-coded in service descriptions, meaning there is a possibility to discover new providers for shipping and payment as well as new actions other than shipping and payment, for example : printing a poster of the book cover.

1.2.2 User Request Processing

The request explaining the user's needs is given by the user to the client. From the user's point of view, he inputs information in the form of text, radio and check box choices, forms etc. This request is combined with a set of QoS-related information about the user (gathered and stored by the client previously) and is used to guide the machine client in the processing of this request. We agree to call this set of information "user profile". For

18. Note that competing in this instance indicates different providers. For example : Amazon and eBay.
19. Each Web service, in this instance, is represented its own set of resources. For example, book selling service is comprised of different sets of resources, each manages a determined source of data : book collections, client orders, shopping carts, etc. The later are also individual resources.

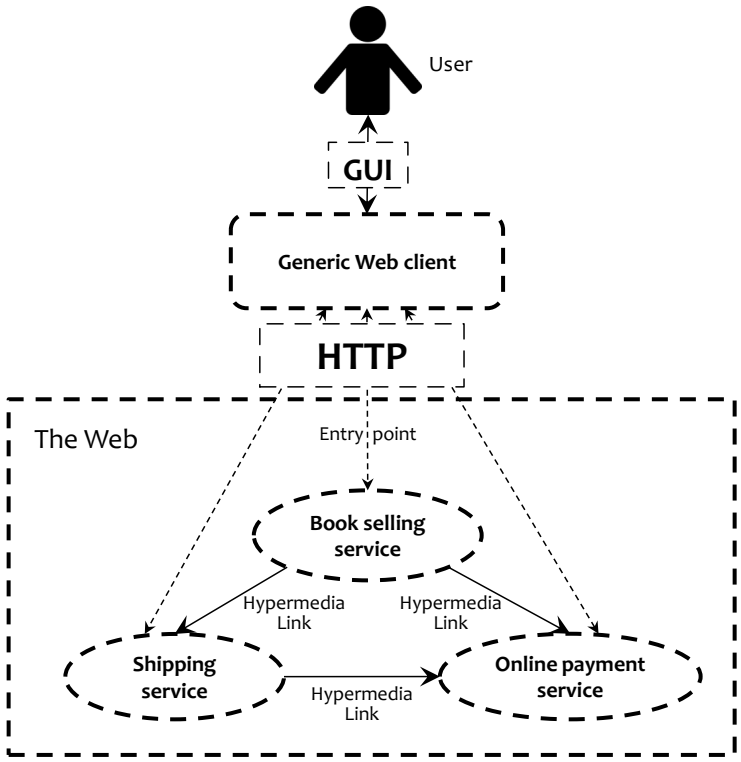

FIGURE 1.1 – Scenario interactions

example, the user may want the services with the best performance and then the best rated among these and does not care about their availability.

A client-side reasoner deduces from the request that it needs to discover a book selling, a shipping service and a payment service. The process of deduction is achieved through a reasoner with a simple subsumption technique. If we take the example of the user's request in the form of a text : "buy book", the reasoner infers that there is a need for a service that sells books in order to allow the user to buy books.

The client begins to crawl the Web looking for the resources needed to answer the user's request. The client needs to discover three services according to the reasoning conducted on the request : the book selling service, the shipping service and the online payment service. The entry point (in this case the book selling service) is either chosen by the user himself when he inputs the request or is obtained by the client using other automated means such as a search engine or by analyzing the user's request history.

1.2.3 Enabling Distributed Affordance

Our objective is to help a user to achieve a complex goal, here buying a book online, with an automated solution. Our work is motivated by the need to enable distributed affordance, which means that the resource discovery process should be automated and hypermedia-driven (consisting in following links between resources).

The advantages of distributed affordance, detailed in [Verborgh et al., 2013], include the possibility of generating opportunities of use for resources while exploring the Web as well as respecting the user preferences. Automating the discovery process can typically be achieved with the help of semantic annotations that help the client decide what are the relevant links to follow.

Building a solution to enable distributed affordance involves several elements :

— **Semantic description of resources :** discovering resources requires them to be semantically described. Semantic annotations provide algorithms with the means to reason about the functionality offered. Such a description should include details about operations available as well as links to other resources according to the HATEOAS principle (cf. section 1.1.3).

— **Resource discovery algorithm :** exploring the Web requires using a scalable discovery algorithm. The latter must make appropriate use of the semantic annotations on resources to automate the process and to optimize the search response time.

The end user should only provide high level objectives to the client, as well as an entry point (URI to start the discovery process). The client should be in charge of interpreting the user request, finding out that buying a book online includes selecting a set of books, choosing a delivery option and paying online. It should explore the Web to discover the ones that help answering the query, orchestrate the interactions and execute the necessary service calls.

1.2.4 Web Resources vs. Classic Web Services

If we were to implement this scenario in the form of classic SOAP-based Web services, we would have the following problems :

1. The use of SOAP-based solutions for Web services would imply the use of XML [20] as a data format. As the Web shifts towards the Linked Data, using JSON-LD [21] as a data format yields more benefits than XML-based formats in Web applications,

20. `https://www.w3.org/TR/xml/`
21. `https://json-ld.org`

since XML-based formats are much more difficult to parse and also because JSON-based formats can be parsed into ready-to-use JavaScript (*the* scripting language for the Web) objects.

2. The discovery and selection of the services would need to go through a centralized repository (UDDI for example) which leads into scalability problems, and slows down the emergence of new providers.
3. The services would have been hard-coded in order to allow specific interactions between different services which hampers the flexibility of the solution when it comes to composing different features from different services, without mentioning the increasing complexity over time and application portability issues.

The use of RESTful Web resources and Linked Data in this scenario solves the previous problems respectively as follows :

1. Using JSON-LD as a data format in the Web of data allows for a full exploitation of the Linked Data and Web of data potential, in addition to the benefits listed above.
2. Discovering and selecting services by exploiting Linked Data advantages allows for a scalable solution and gives the chance for new providers to emerge if their services are enticing.
3. RESTful APIs are developed for no specific application and their adoption enables the use of generic clients since the resources themselves have uniform interfaces by definition. RDF descriptions of operations, inputs and outputs ease the composition of different resources while avoiding incompatibility problems thanks to the loose coupling property of the REST architectural style.

1.3 Research Problems

In this section we identify and explain the key scientific locks we try to address in our work. These problems have been identified since the early stages of adopting service technologies on the Web at large [Hansen et al., 2002]. We identify a set of problems with the setup illustrated by the scenario in the previous section :

First, the client only disposes of the entry point at the beginning. There is a need to enable the client to crawl through the services related to the entry point and then the services related to them and so on. As we are working in the context of RESTful Web services, the HATEOAS principle (cf. section 1.1.3) applies to all services the client accesses. This means every time the client accesses a resource, the later should indicate

the links to the next accessible resources (the next application state). The client needs to crawl the Web in order to find services that can provide the functionality needed to answer the user's request and has no other way of accessing them [22]. The distributed affordance concept also states that actions made possible by the application state may not be enough to answer the user's need, therefore the need to provide more actions to the user by exploring the Web has to be satisfied.

Secondly, the client may find multiple services that fulfil each functionality and has to choose one based on the user's QoS requirements. It may also find only services that match with the functionality but does not match with the user's QoS requirements.

Finally the client needs to be able to automatically and dynamically compose the discovered and selected services. The invocation of the involved services needs to be done following a certain workflow for the sake of delivering the correct result to the user. The client also needs to know what are the different data formats each service requests and delivers, as well as their meaning.

As a result of the changes to the way we work with Web services and the emergence of Linked Data, the typical approaches to discover, compose, orchestrate and utilize linked services on the Web require complete overhauling of existing technologies in order to harness the full benefits provided by the REST principles, the semantic Web and Linked Data. Reaping the benefits from these advances requires adapting the way information systems and applications interact with the Web. Using semantic Web advances to automate resource discovery and composition is a recent topic and has only been explored by few works [Kovatsch et al., 2015]. Some approaches extend BPEL with RESTful support [Pautasso, 2009a], some focus on the semantic description of resources to drive the discovery and composition algorithms [Alarcón et al., 2010] and some use reasoning techniques to automate the resolution of user queries [Verborgh et al., 2011a].

1.3.1 Description

Describing a service consists in adding descriptive data elements to the service representation. We define semantic description of a service as the act of annotating its data elements and descriptions with machine-readable semantic information. The data elements in question include business-level information (including potentially non-functional properties), service behaviour (interactions) and links to other services or resources. Web service description is, in our view, the most important aspect to consider in the design of any Web Service oriented architecture. The automation of service discovery, selection and

22. Except maybe through the user's history or a search engine, but that remains unreliable.

composition on the Web cannot be achieved without properly described and semantically annotated services. In order to exploit the full potential of Linked Web services, there is a need for a detailed semantic description about its behaviour as well as its interlinking. The semantic description of a service represents all the important information to identify the nature of the service's inputs and outputs as well as the way it processes data. Many efforts have been made by researchers to establish semantic description models for services on the Web [Kopecký et al., 2008] [Alarcon and Wilde, 2010] [Verborgh et al., 2011b] [John and Rajasree, 2013].

Resolving the problem of description revolves around constructing a generic set of information compliant with Semantic Web and REST principles. The main purpose of this information is to facilitate the automation of interactions with Web clients. The exploitation of this semantic description allows the clients to make more complex interactions with these services and allow the construction of composite services that enable value-added applications. Having a complete semantic description of linked Web services allows dynamic and automatic answering to complex user requests that need to involve multiple services at once. Semantic descriptions guide the discovery process and help clients decide what is the next service to explore in order to look for the remaining features required by the user's request. It also contains vital information about Quality of Service attributes, which is a major deciding element in the selection process. In the case where we discover multiple services that can fulfil a required task within the user's request (cf. section 1.3.3), this descriptive information about non-functional properties helps make the decision regarding what services to choose. It also contains the information about input and output of the services for every available operation in order to facilitate the invocation and the composition of multiple services. One of the main challenges is to establish a trade-off between the expressivity and complexity of descriptions in terms of data and computations.

1.3.2 Discovery

Discovering consists in browsing or crawling a defined set of resources, often the Web, in search for services capable of performing a desired action or task. Multiple service candidates can be discovered for each action or task. As centralized solutions for service discovery have proven not to scale well [Anadiotis et al., 2009a], the need for distributed service discovery has emerged [Verborgh et al., 2013]. The discovery of services that fulfil a certain task in the process of answering the user's request also brings in the need for selecting the most suitable of these candidates to actually execute the task needed. Since Web application platforms have shifted from classic RPC-oriented Web services

Task 1		Task 2		
URI1	Operation1	URI3	Operation3	...
URI2	Operation2	URI4	Operation4	
...		...		

FIGURE 1.2 – Discovery Algorithm Results

into linked Web services and RESTful APIs, researchers have been focusing on adapting existing discovery solutions to this new architecture [Anadiotis et al., 2009b] [Maamar et al., 2011a].

The discovery problem in the context of Linked Web services can be formulated as a directed labeled (sometimes weighted) graph search problem. The graph in question is the Web. The nodes are the different services interlinked by semantically annotated RDF links which constitute the edges of the graph. Resolving the discovery problem amounts to constructing the algorithm that crawls through the Web starting from the entry point given by the user and following the annotated links available on each node in order to find at least one successful match for every task required by the user's request. A match between a task and a service (node) is successful if the service provides an operation that can successfully perform the process required by the task and returns the desired results. The result of the algorithm (namely, the solution to a given discovery problem) is a list containing the different tasks and the URIs that point to services as well as the specific operation within the service that actually performs the task in question. The results of the algorithm are illustrated in Fig 1.2. The different URIs identify the services discovered and the operations describe the exact operation to call. Other parameters related to inputs/outputs, which are not illustrated, may also need to be specified.

1.3.3 Selection

Selection is the action of choosing a subset of candidates, from the entire set of services discovered that complies with the discovery criteria. Selection is often done on the basis of its own criteria such as QoS, non-functional properties, user requirements and preferences etc. The Web contains a variety services that provide similar services but with a varying degree of quality. Also, the user's needs may vary depending on many variables including the user's location, language, personal preferences etc. This is why having a clear and complete description of services and their operations as well as their QoS properties is

Task 1		Task 2		...
URI x	Operation a	URI y	Operation b	

FIGURE 1.3 – Selection Algorithm Results

paramount. Finding the optimal solution for the selection problem with multiple QoS constraints is a NP-hard combinatorial optimization problem [Yu et al., 2007]. There are many efforts made by the research community in order to propose solutions for this problem, which resulted in a wide-variety of algorithms and meta-heuristics that give approximately the best solutions [Alrifai et al., 2010] [Wang et al., 2011] [Zhao et al., 2014].

The discovery algorithm returns several candidate Web services for every task needed to answer the user's request. Finding a solution amounts to select the best candidate for each task in order to obtain the highest overall QoS and satisfying the user's specific constraints. In other words, we need an algorithm that takes the discovery process results as input and returns a table with only one service selected for each task. The table in question illustrated in Fig 1.3. The same remarks explained at the end of section 1.3.2 apply here.

In our work we follow a different setup where the search space of candidates is progressively discovered by following links between resources, rather than having the entire discovery result in one iteration, which presents new challenges to overcome.

1.3.4 Composition

Composition can be defined as the process of invoking collaboratively and combining inputs and outputs of more than one service in order to achieve an end-goal not possible to achieve otherwise with only one service. We talk about value-added applications. Despite the evolution of service technologies, the need for service composition to build complex applications is still present because of the distributed nature of the Web. However, the challenges we have to overcome have changed. Many notable research efforts have been made in this area [Pautasso, 2009b] [Zhao and Doshi, 2009] [Stadtmüller, 2012] [Pedrinaci et al., 2014a].

The interlinked nature of linked Web services opens more opportunities to discover related services that can potentially contribute to solving the user's request. However, as linked RESTful services rely on uniform interfaces, stateless interactions and loose

coupling between client and services, composing linked services is different than in classic SOA services. The composition engine in a resource-centric architecture is the client itself since it is responsible for carrying out the transition in the application state according to the HATEOAS constraint, which offers a great deal of flexibility, adaptability and robustness [Stadtmüller, 2012]. The challenges lie in the full exploitation of the links between services and their semantics to automate the composition process and eliminate manual human intervention.

Furthermore, the diversity of data formats RESTful services exchange in the Web of Data has to be taken into account by the client, which takes the responsibility of consolidating and integrating the data from the different services involved in the composition. Enabling distributed affordance in our approach opens more composition opportunities that may be interesting for the user. Instead of relying on statically generated, inflexible and hard-coded service mashups, we need to dynamically aggregate and "mash" together functionality and data from the discovered and selected services. Storing composition mashups for future reuse and eventual sharing in a social-like model is also an important challenge we aim to address.

1.3.5 Contribution summary

This thesis proposes the following solutions to address the aforementioned problems.

Firstly, we propose a *model for semantic description of Web resources* that incorporates semantic annotations over links, operations, QoS-related data and other business-level descriptive elements. This model provides a clear description of the nature of the links to the related resources and what to expect if the client decides to follow them as well as clear details about functionality and features provided by the allowed operations on the resource, the data inputs/outputs and also the error codes.

Secondly, we propose a *discovery algorithm* that exploits the semantic annotations over the links present in the resource descriptions in order to search the most relevant services needed to answer the user's request. The identification of a suitable service is driven with the help the annotations over its operations and the other informative functional semantics present in the descriptions.

Thirdly, we propose a *selection algorithm* that uses both the user QoS preferences (in the form of a user profile) and the QoS-related information in the resource descriptions in order to perform the selection of the most suitable resource for a specific task in the user's request. The advantage of this algorithm is that it is executed in parallel with the discovery process while it is crawling through the Web.

Lastly, we show how composition can be achieved with the help of the aforementioned description model and algorithms and we propose a solution that allows users to *store*, *reuse* and *share* flexible and dynamically generated composition workflows/mashups that perform specific complex tasks otherwise impossible to perform with a single service. Services part of a specific stored workflow can be replaced dynamically, if unavailable at the time of the execution for example, without changing the whole process as long as it offers similar functionality and its inputs/outputs are compatible.

1.4 Document Organization

The rest of this thesis is structured as follows :

Chapter 2 addresses the description problem. We present and analyse the different efforts undertaken by the research community to describe RESTful linked Web services. We also discuss the need for descriptions in context of today's Web evolution. Finally, we present and discuss our contribution, namely the *Descriptor* concept.

Chapter 3 addresses the discovery and selection problems. The problems of discovery and selection are closely related. For the sake of clarity we have decided to treat the two problems in one chapter. We start the chapter by presenting a state of the art of the discovery and selection of RESTful linked Web services and discussing the important works led in this domain. We also discuss the important role of the description in driving the discovery and selection processes. Next, we present and discuss our contribution for the discovery problem, namely a BFS-based algorithm that relies on the semantic annotations in the description. Finally, we discuss the different setups possible for the selection process and present our contribution for the selection problem, namely an algorithm that is executed on the fly while the discovery process is running.

Chapter 4 addresses the composition problem. We start by presenting the different approaches adopted by the research community to resolve the problem. We also enumerate the different challenges that arise and show how descriptors and the discovery/selection algorithms play a role in facilitating the composition. Finally, we present and discuss our contribution, namely the *Composition Directories*.

Chapter 5 fences this document with a conclusion. We summarize the research problems raised in our work, discuss our contributions and how they collectively address the problems and finally show possible avenues for future research.

Chapter 2

Semantic Description of RESTful Linked Services

Contents

In this chapter, we present an approach to describe services on the Web in the context of Linked Data and the REST architectural style. Our contribution is a data structure represented by an RDF document (using the concrete syntax of JSON-LD) containing the interaction possibilities a RESTful service offers alongside the available semantically annotated links to other related services. We call it a *descriptor*. The description model proposed here supports the contributions presented in chapter 3 and chapter 4.

2.1 Introduction

During the last decade, the emergence of Web services has been a major success to enable data interoperability on the Web. In parallel, the advent of Linked Data and the adoption of its principles for data publishing on the Web has standardized the models and formats Web services produce and consume on the data level. However, on the service level, models for descriptions have yet to reach the level of maturity required for a large scale such as the Web. Service description either rely on obsolete formats that are not adapted to the nature of the data on the Web of Data or on heterogeneous description

models that require significant efforts to enable interoperability on the service level. This hampered the rise of standard solutions for a scalable discovery and an automatic composition. Moreover, the available description formats suffer from the lack of standardized and explicit semantics that are required by the principles of Linked Data and focus each on a specific aspect of service description while neglecting the other aspects.

2.1.1 REST and Service Description Models

The rise of the REST architectural style as a dominant paradigm for the design and implementation of Web service APIs can not be denied. The adoption of RESTful services changes the way services are designed, implemented and described. The typical API built around functions and input/output parameters is slowly being abandoned. The management of the application state, which was usually handled server-side, is now at the charge of the client software (the browser or the application).

During the last few years, the typical Web services relying on the XML service technology stack (SOAP, WSDL, UDDI) is slowly being abandoned for the profit of REST-based approaches. The success of RESTful Web services is highlighted via Web sites such as ProgrammableWeb [23] that counted 2125 SOAP-based APIs versus 6833 REST-based ones in 2013, and more than 18000 APIs in 2017 including almost 15000 RESTful APIs which represent roughly 81% of the total number of APIs. The rise in popularity of RESTful Web services is regarded as a huge boon for the research community and especially for research activity about service description. The constraints imposed by REST directly impact how a service should be described. On a formal level, the constraints relevant to this context are the following [Fielding, 2000] :

— Resource identification : a service is exposed as a set of one or more Web resources each one of them has its own unique identifier.
— Uniform interface : the interaction possibilities with exposed resources are well-determined and the same for all resources.
— Self-descriptive messages : the messages exchanged during interactions should imply (implicitly or explicitly) their format and how it should be parsed.
— Hypermedia driven application state : the messages may contain typed hypermedia links that guide the interactions with the service.

Concretely, these constraints have led to the adoption of the following Web standard technologies for service design and implementation, respectively :

— URIs for resource identification.

23. `https://www.programmableweb.com/api-research`

— HTTP as a uniform interface for message transfer and resource manipulation.
— the standard serialization formats such as XML, JSON and HTML as message formats.
— Hyperlinks as a mechanism to allow a hypermedia driven application state.

The norm for service description on the *syntactic* level has moved towards this stack of technologies, which have been adopted as the standard by normalization organizations such as W3C [24] and IETF [25].

2.1.2 Description Models on the Semantic Level

Recent advances in the semantic Web research area have been promoting Linked Data [Bizer et al., 2009a] and a set of languages and tools such as JSON-LD [Sporny et al., 2014], RDF [Lassila and Swick, 1999], OWL [Schreiber and Dean, 2004], SPARQL [Prud'hommeaux and Seaborne, 2008] and POWDER [Archer et al., 2009], that allow to annotate Web data, resources and services with explicit, machine-readable semantics that can be utilized in conjunction with advanced reasoning mechanisms and similarity interlinking to connect resources to each other in a way that allows advanced and automated interactions.

On the *semantic* level there is still a lot of heterogeneity in service description formats. This is mainly due to each solution relying on its own vocabulary or ontology to annotate the description elements of services. This unwanted diversity in vocabularies for service description was the result of the openness of the Web of Data to different models. In theory, this openness allows services to be described using the format most adequate for them, but in practice this leads service consumers to deal with all the various models and tools required to interact with these services. This means that clients have to expect heterogeneity in data models and service descriptions by design complicating their implementation [Wilde, 2010a]. Although Linked Data relies on RDF for data models, including service descriptions, RDF is not a specific data model, but rather a data metamodel, or a model for data models.

Gradually, the efforts to describe services started converging towards the reuse of popular vocabularies of solutions that showed promising results, but until now no data model has been adopted as the standard for RESTful service descriptions. Moreover, there was little to no backward compatibility with services that relied on classic service description formats and there was no support for incremental annotation of the service

24. World Wide Web Consortium : `https://www.w3.org`
25. Internet Engineering Task Force : `https://ietf.org`

data elements.

2.1.3 Sketching the Ideal Description Model

Reaping the benefits from the advances of semantic Web and service orientation requires adapting the way information systems and applications interact with the Web. The first step is to establish an efficient description model for linked services in order to facilitate the discovery and composition and provide the information needed for the service to interact with external entities automatically, namely Web clients.

In this context, the description model must provide useful information to the Web clients at different points in time :

— When searching a service that provides a certain functionality ; we are talking about discovery.
— When choosing one amongst many services that provide the same functionality ; we are talking about selection.
— When requesting the service to actually perform the functionality needed to answer the user request ; we are talking about invocation in the context of a composition.

The way we see it, service description needs to take various aspects into account to enable automatic discovery, selection and composition :

— Business data aspect : clients need to know the format and semantics of the data they exchange with the service.
— Operation aspect : clients need to know how the behaviour of a service when they call a specific operation provided by it.
— Interlinking aspect : clients need to be fed with links to other resources so they can continue to advance the application state.
— Non-functional aspect : clients need to know the non-functional and QoS properties of the service so that they can decide whether or not to interact with it.

The description of these aspects needs to be explicit and expressive enough to achieve the level of automation desired, but at the same time we have to take into account the complexity of the descriptions at the calculation and data level. In short, there is a trade-off to make between the expressivity and complexity of descriptions.

2.1.4 Ramifications of Description for Service Consumers

Having a complete description of services allows flexible answering to complex user requests that need to involve multiple services at once. Descriptions guide the discovery

process and help clients decide what is the next service to explore in order to look for the remaining services required by the user's request. It also contains vital information about Quality of Service, which is a major deciding element in the selection process. In the case where we discover multiple services that can fulfil a required task, this descriptive information about service performance, cost, etc. help make the decision regarding what service to choose. It also contains the information about the behaviour, inputs and outputs of operations in order to facilitate the invocation and the composition of multiple services.

Regrouping all this information into one data structure, that is associated with the resource it describes, trivializes the access to descriptions for clients and simplifies their design and implementation.

2.1.5 Contribution Summary

In this chapter, we propose a solution to describe linked services based on REST and Linked Data principles. We dub our proposal the *descriptor* mechanism. It is a data structure based on the Hydra core vocabulary [Lanthaler and Guetl, 2013] and using JSON-LD [Sporny et al., 2014] as a serialization format. JSON-LD is a concrete RDF syntax, meaning that descriptors are formally RDF graphs.

Descriptors are a mechanism to describe RESTful linked Web services that complies with the practices of Linked Data and semantic Web and achieves semantic interoperability on the data and service level on the Web scale. Descriptors are the first step in the integration of service technologies on the semantic Web.

2.2 Related Work : Description

Description of Web services has been an active research field since the introduction of the service paradigm into the World Wide Web. Researchers have been particularly interested by the description of service API functionality and typing their inputs and outputs. Until recently, there was not much effort put into describing the semantics of the operations and data let alone linking and describing links to other services or resources. The advent of the semantic Web technologies as well as Linked Data and the Web of data has shed light on the need to semantically annotate various aspects of services.

Many efforts have been made in this regard especially annotating data elements that services manipulate. This was a necessary step forward in order to allow the integration of service practices on the semantic Web and allow services to consume the wealth of information on the Web of data that remained, until recently, unexploited. The annotation

of the data services produce and consume means that services can now consume and produce Linked Data. In other words, Web services can now benefit from and contribute to the linked information space of the Web of Data. We talk about linked services (cf. section 1.1.1), services described with Linked Data that produce and consume Linked Data [Pedrinaci et al., 2010a].

Meanwhile, efforts to annotate linked service operations and interlinking have been sparse. This is made more apparent by the fact that more complex service manipulations such as discovery and composition for linked services are still made manually or without enough automation. On one hand, discovery of linked services relies either on centralized repositories (a la UDDI) which come with their fair share of drawbacks [Anadiotis et al., 2009c] most notable of which is being a SPoF [26] or on general-purpose search engines which do not yield satisfactory results and are unreliable mainly because they rely on keyword search rather than semantic terms and do not incorporate search parameters specific to Web service discovery or filtering [Hatzi et al., 2012]. On the other hand, composition of services is still made using hard-coded, rigid and inflexible mashups.

This, combined with the success of the Web of Data [Pedrinaci et al., 2010b], has sparked the need for semantic descriptions of service operations, links and other non-functional properties such as QoS. The solutions proposed in this area have been lacking however. They either focus on one aspect and fail to describe the other aspects or fail to integrate with the practices of semantic Web and Linked Data. The increasing popularity of REST APIs also meant that service descriptions must adhere to REST constraints, mainly self-descriptive messages, stateless interactions and hypermedia driven application state.

In this section we present a detailed study of different existing efforts that attempt to solve the problem of description in the context of Linked Data and the REST architecture. We categorize these existing related works in a pseudo-chronological classification.

2.2.1 State of the Art of Service Description

Although a lot of research has been conducted in the field of RESTful linked service description [Lanthaler et al., 2010] [Lanthaler and Gütl, 2010] [M'Barek and Tata, 2008], and even though some of them are W3C recommendations, none of the proposed solutions gained enough traction, and their widespread amongst Web service providers has been limited by many factors [Verborgh et al., 2011b]. In fact, most description solutions completely ignore the HATEOAS constraint of REST which makes those descriptions not

26. Single Point of Failure

adapted to enable automatic discovery and composition. Some of the efforts violate other REST constraints and introduce unnecessary coupling between clients and servers.

The earlier research activity yielded solutions that are considered too verbose and heavyweight for the lightweight approach of RESTful Web APIs. In addition to that the earlier formats focused on the description of inputs, outputs, data types and exceptions which are typical to classic RPC-oriented services, while the resource and hypermedia-oriented REST architecture is centred around functionality and links.

In the following, we propose a classification of existing related efforts to describe RESTful linked services. As we see it, these works can be categorized as follows :

2.2.1.1 Syntactic Description Solutions for Classic Web Services

The first solutions offered for describing the classic RPC-based Web services offer no semantics in descriptions, and thus no automatic service manipulation can be done with service described like this. The most notorious solution and the de-facto description language for classic Web services in the WS-* technology stack is WSDL.

WSDL

Web Services Description Language [27] [Christensen et al., 2001] or WSDL is an XML description language for Web services based on the XML protocol SOAP. It is the main description language for classic RPC based Web service technology stack. It describes Web services from a RPC message-oriented point of view. It is purely a syntactic description of service interfaces, operations, and inputs/outputs and does not convey any semantics of the service it describes.

Although WSDL has the word "'Web"' in the name, it is hardly adapted to the architecture of the Web since it is RPC-oriented and thus incompatible with the resource oriented nature of the Web today, where REST offers more advantages such as scalability and Linked Data offers semantic interoperability. It is optimized for closed-world systems, typically adopted in the enterprise world, and offers little scalability for an open information space such as the World Wide Web. It is also considered too verbose and inflexible to change, since a lot of the element it describes are already fixed by the REST constraints. Also, the RPC paradigm that uses SOAP as a protocol for message exchange does not fully use the power of HTTP because the latter is only a tunnelling protocol for SOAP messages, which is reflected in the WSDL descriptions.

27. `https://www.w3.org/TR/2001/NOTE-wsdl-20010315`

2.2.1.2 Semantic Description Solutions for Classic Web Services

After the advent of the semantic Web, the huge potential of semantically annotated Web service descriptions became apparent. These solutions offer a level of semantic description to different service aspects in order to support automatic processing of these descriptions by clients or agents so that tasks such as discovery and composition can be automated. They are still targeted towards RPC-oriented classic Web services, although some can be used to describe REST services with considerable efforts, and most of them build on WSDL in order to add semantic annotations on its descriptive elements or try to alleviate some of its inherent limitations in the semantic Web environment. They are considered too heavyweight and complex for the REST's lightweight resource-oriented approach and do not support scalability on the Web.

SAWSDL

Semantic Annotations for WSDL and XML Schema [28] [Kopeckỳ et al., 2007] or SAWSDL is a description language that allows adding semantics to WSDL description components such as interfaces, operations and inputs/outputs. SAWSDL is thus an incremental approach at semantically annotating WSDL descriptions.

SAWSDL defines how semantic annotations are added using references to ontologies and other semantic models without specifying the language to represent them, but rather how to reference the concepts from within the descriptions. This is achieved thanks to two extension attribute types : *modelReference* which allows multiple semantic annotations to be associated with a WSDL or XML Schema component and *liftingSchemaMapping /loweringSchemaMapping* which allow to specify a mapping between the descriptive XML elements and a semantic model.

One of the main drawbacks of SAWSDL are its annotations, which are aimed at discovery in a central registry such as UDDI and not a scalable information space such as the Web. Also, SAWSDL does not support defining non-functional properties and does not specify any specific formal semantics to annotate the service description elements. Although it brings machine-readable semantic annotations, it still inherits the disadvantages from WSDL and is thus considered to be inflexible to change and complex.

OWL-S

OWL-S [29] [Martin et al., 2004] is a semantic ontology based on the W3C standard

28. https://www.w3.org/TR/sawsdl/
29. https://www.w3.org/Submission/OWL-S/

OWL [30] for describing semantic Web services in RDF. It was designed to allow the discovery, invocation and composition of services with automation in mind.

OWL-S distinguishes three separate description aspects : service *profile*, service *model* and service *grounding*. In a nutshell, the service *profile* is the primary information used to advertise and discover the service being described, the service *model* is the primary information about how the service operates and is used by clients to interact with the service and the service *grounding* describes the way the service communicates and how it interoperates. In other words, there is a separation between the descriptions aimed at interaction from descriptions aimed to enable discovery.

The usability of the semantic annotations added to OWL-S descriptions is unfortunately too low to achieve a level of automation of discovery and composition required by today's Web. OWL-S also has a considerable level of perceived complexity for the lightweight approach of RESTful services. There is also little support for interlinking in OWL-S descriptions which is not enough to enforce REST's HATEOAS constraint in described services. In addition to that, OWL-S does not impose constraint for the coherence on some of its description aspects, notably service *profile* and service *model*.

WSMO

Web Service Modeling Ontology [31] [Roman et al., 2005] or WSMO provides a formal language to semantically describe various Web service aspects. It has been designed in order to exploit the semantic annotations in service descriptions and automatically discover and compose services on the Web.

WSMO identifies four top-level elements as the main concepts that have to be described in order to define a semantic Web service description :

— *Ontologies* provide machine-readable formal definitions of other WSMO description elements and the domain-related data.
— *Web services* represent the actual Web service descriptions of interfaces, capabilities and internal functioning of the service entities.
— *Goals* describes aspects related to the requested functionality while taking into account the user's end-goal and desires.
— *Mediators* handle the interoperability concerns by proposing mappings to resolve heterogeneities at the data, process and protocol levels.

WSMO allows the discovery of Web services based on a goal-oriented approach, meaning that starting from a high level user's request, the client can automatically discover

30. Web Ontology Language : `https://www.w3.org/OWL/`
31. `https://www.w3.org/Submission/WSMO/`

WSMO-described services based on the elemental goals in that user's request.

Despite the fact that WSMO offers a comprehensive framework for describing various aspects of Web services, the main critics of WSMO remain its sheer complexity and sizeable descriptions. Also, WSMO lacks explicit semantic descriptions for the actual operations that have to be called by clients in order to execute the desired functionality. In addition to that, WSMO has been developed in isolation from the standards established by W3C as mentioned by the comment on the sumbission in [Bournez, 2005]. A thorough analysis and critical evaluation of WSMO can be found here [Sharifi and Bayram, 2016].

2.2.1.3 Description Solutions for Classic Web Services Adapted to the REST Architectural Style

After it became apparent that description mechanisms for RPC-oriented services were not adapted to the resource-oriented REST architecture of the Web, researchers started tailoring the description models to directly support the REST constraints and trying to move from the input-operation-output description format towards a more resource-oriented and REST-friendly format of resources/functionality/links. However, the fact that these solutions had to be able to describe classic Web services as well was a double-edged sword. On the one-hand, it enabled a unified approach to describe, discover and compose services regardless of their underlying architectural design, but on the other hand, they remained complex and inflexible compared to the lightweight approach of the REST architecture.

WSDL 2.0

WSDL 2.0 [32] [Chinnici et al., 2007] is a description language for services that tries to address the problems of WSDL and support the description of REST services. It maps the HTTP methods available in RESTful resources to the services being described. WSDL 2.0 provides a unified description format for both RPC-based services and REST-based services irrespective of their underlying architectural design.

Although WSDL 2.0 allows the description of REST services, it does so in an RPC-oriented approach still and thus eclipses a lot of the properties offered by the REST architecture. It also still suffer from some of the problems as WSDL, namely complexity and lack of meaningful semantic annotations.

WSMO-Lite

32. `https://www.w3.org/TR/wsdl20/`

WSMO-Lite [33] [Vitvar et al., 2008] is an effort that defines a service description ontology and uses the SAWSDL description language as an annotation mechanism for Web service descriptions. It builds an incremental layer on top of existing Web service descriptions.

WSMO-Lite was designed with regards to the W3C standards established at the time and the need for a lightweight service ontology. It adopts WSMO's semantics and makes its semantics lighter by allowing the use of ontology languages with RDF syntax. WSMO-Lite distinguishes the following service description elements :

- *Information Model* : defines the data model for inputs, outputs and fault messages.
- *Functional Description* : defines what a service can offer to its clients when it is invoked, in other words service functionality.
- *Non-Functional Description* : defines any incidental details specific to a service provider, or the service implementation or its running environment.
- *Behavioural Description* : defines external (public choreography) and internal (private workflow) behaviour.
- *Technical Description* : defines messaging details, such as message serializations, communication protocols, and physical service access points.

WSMO-Lite ontology concepts are reused by many of the relatively recent efforts at proposing a description mechanism for RESTful linked Web services. Contrary to WSMO, the goal-oriented discovery and Behavioural descriptions are made implicit in WSMO-Lite, in order to reduce the complexity of its description model.

Linked USDL

Linked Unified Service Description Language [Pedrinaci et al., 2014a] or Linked USDL is a vocabulary for capturing and sharing service descriptions aimed at supporting service trading on the Web. This work focuses on the technical, operational, economic, social, legal and business contexts of services to enable multiple cross-domain services from various service providers to interoperate.

Linked USDL builds upon USDL [34], which the authors considered too complex and inflexible to extensions, and combine it with Web-centric technologies such as semantic Web and Linked Data in order to support an open and automatic trading of potentially heterogeneous services on the Web scale. The model uses URIs for service and resource identification, links to point to related resource and complementary services and HTTP as an interaction protocol with services. It promotes the reuse of formal ontology repre-

33. https://www.w3.org/Submission/WSMO-Lite/
34. https://www.w3.org/2005/Incubator/usdl/

sentation languages and widely used vocabularies to capture service semantics according to Linked Data practices. It captures service interaction interfaces, pricing models, service level agreements and related legal issues.

While the aim of Linked USDL is to provide a good level of interoperation between highly heterogeneous cross-domain services, it imposes on service providers the use of vocabularies, albeit popular and widely used, to capture the semantics of domain-specific data. Despite its aim to reduce USDL's complexity, we still think that Linked USDL descriptions are rather complex for Web services in today's lightweight-friendly resource-oriented Web.

2.2.1.4 Description Solutions for REST Services

Researchers started realizing that unified description efforts were not the ideal solution to describe REST services. With the rapid rise in popularity of RESTful APIs and the decline of classic Web services, new solutions specifically tailored towards capturing descriptive aspects of REST services while recognizing the REST constraints started emerging. These solutions still suffer from the verbosity and fail to adopt the lightweight extensible approach advocated by the combination of REST architecture with Linked Data and semantic Web practices.

WADL

Web Application Description Language[35] [Hadley, 2006] or WADL is an XML-based syntactic description language specifically designed to describe REST services that aims at the proper usage of HTTP as a transfer protocol for messages. Resources, which are identified by predefined URI patterns, are at the center of descriptions and WADL captures the relationship between them. Service interfaces are modelled in descriptions by constructing requests containing the HTTP method, the inputs, outputs and response status code.

Although WADL is aimed at the description of REST services, most of the time it violates REST constraints especially hypermedia driven application state and introduces unnecessary coupling between clients and servers with the predefined URI patterns for resource and input/output parameters, which imposes constraints on server URI defining schemes. Also, even though it is aimed at describing lightweight REST services, WADL descriptions are considered complex and require a lot of effort and knowledge to establish.

35. `https://www.w3.org/Submission/wadl/`

hRESTS

HTML format for describing RESTful Services [Kopeckỳ et al., 2008] or hRESTS is a microformat for labeling Web service APIs with machine-readable descriptions. It describes the main aspects of a service interface such as the operations and their inputs/outputs. hRESTS was primarily designed to enrich existing service descriptions, in the form of human-readable documentation, in HTML.

hRESTS' human-first approach is tailored towards assisting Web service designers and developers that use these services to construct mashups and integrate the services in more advanced hardcoded applications, and thus is not better suited for generic machine clients that perform tasks such as automatic discovery and composition. In fact, although hRESTS describes service interfaces and operations with machine-readable information, other aspects of the service such as the exchanged data schemas and inputs/ouputs are not explicitly described semantically, there is only labels containing human-readable descriptions on these elements.

SA-REST

Semantic Annotation for REST [36] [Lathem et al., 2007] or SA-REST is a format that allows to add additional metadata to REST APIs descriptions in HTML. It provides a list of input and output parameters, methods and URIs exposed by a REST service by means of property-value pairs or RDFa annotations. SA-REST allows to add semantic information to descriptions from ontologies and other semantic models in order to facilitate the discovery of the services and their composition not only by humans, which can do so using natural language descriptions and documentation, but also by machine-clients, which exploit the semantic annotations on these human-readable documentations to automate the discovery and composition processes.

SA-REST lacks support for the interoperability of heterogeneous services, especially on the data level. Also, even though the main reason behind developing SA-REST was to provide semantic metadata to service descriptions, the machine-processable annotations do not support a truly automated discovery and composition without humans intervention in the form of manual mashup creation and manual service retrieval.

MicroWSMO

MicroWSMO [37] [Kopecky and Vitvar, 2008] is a service ontology extension that builds on hRESTS to add semantic annotations to service descriptions and documentation

36. https://www.w3.org/Submission/SA-REST/
37. http://www.wsmo.org/TR/d38/v0.1/20080219/

in HTML. It was specifically designed to describe RESTful services with the REST constraints and automation of discovery and composition in mind. Its main purpose is to extend hRESTS with semantic annotations to exchanged data and inputs/outputs.

MicroWSMO focuses on the description of semantics of the HTTP operations, especially POST, since it does not have explicit semantics in the HTTP specification, and it also focuses on semantically annotating data formats used as inputs and outputs in resource representations. The structure of descriptions in MicroWSMO is similar to the one used in WSMO-Lite but with several differences due to the constraints imposed by REST which simplify some aspects :

— *Information model* : represents data in resource representations using a domain-related ontology, namely inputs and outputs.
— *Functional semantics* : specify the functionality provided by the service and particularly HTTP method POST.
— *Behavioural descriptions* : describe the hypermedia structure of the resources making the Web service.
— *Technical descriptions* : this is generally the service URI, since other technical details are fixed by REST's constraints.
— *Non-Functional Description* : service policies or other details specific to the implementation or running environment of a service.

Additionally MicroWSMO specifies grounding of these annotations to the data elements in resource representations in the form of lowering/lifting mappings. As mentioned in [Lanthaler et al., 2010], MircoWSMO and SAWSDL both can use WSMO-Lite ontology semantics which allows service manipulations such as discovery and composition to be done independently from the underlying service implementation [Maleshkova et al., 2009].

2.2.1.5 Lightweight Semantic Description Solutions for REST Services

After is became apparent that lightweight extensible approaches for describing RESTful Web services were superior to heavyweight ones because REST defines and fixes a lot of descriptions by design, the research community started to move towards more lightweight description models with reuse of existing vocabularies for semantic annotations rather than inventing new ones. Also the convergence of semantic Web technologies with REST architecture has led to the adoption of Linked Data principles in description solutions.

Siren

The Siren project [38] is a hypermedia specification using the JSON format [39] for describing entities for Web APIs similarly to how HTML represents documents on a Web site. It supports a resource-oriented design style which is well adapted to RESTful Web APIs. Using Siren allows to easily provide a action-based interface through Web APIs with support for hypermedia.

Siren offers a variety of structures to describe the elements within a Web API. An entity represents a URI-addressable resource. Each entity has a description used to communicate information about the entity itself and the data it contains. This description includes a `title` field that textually describes the entity in question, a `class` field that describes the concept of the entity (for example `book`), a `properties` field that contains a set of key-value pairs describing the entity and an `entities` field that contains a list of related sub-entities. An entity has also an `action` field that lists a set of available actions on the current entity, described with the corresponding HTTP method as well as a `links` field that lists items containing navigational links including link to itself.

Siren was the first format we used to inspire our description model in [Bennara et al., 2014a]. Although Siren offers a good resource-oriented and hypermedia-based description with its descriptive elements, using it revealed its limitations in the context of our research. First it does not support explicit semantics of its descriptive data elements, all that is provided is textual descriptions in human language for Web developers and no machine-readable semantics are given. It also uses JSON as a serialization format instead of JSON-LD, that supports Linked Data principles and allows the use of vocabularies and name spaces to describe properties with semantic annotations. Also, although extensible, it does not propose a way to describe non-functional properties of Web APIs.

RESTdoc

RESTdoc [John and Rajasree, 2013] is a description solution that combines multiple micro-formats in order to semantically describe RESTful service elements in HTML using the JSON format. It uses RDFS annotations in order to enable interlinking of RESTful resources and offers the possibility of converting the descriptions into RDF and vice-versa.

RESTdoc reuses the special purpose annotations provided by HTML and JavaScript in order to annotate the service elements. It describes elements such as service name, URI and a set of annotated attributes/properties. It offers also a discovery mechanism that uses HTML Link element on a Web resource in order to point to other resource descriptions. Composition is achieved by constructing an RDF graph of converted descriptions

38. urlhttps ://github.com/kevinswiber/siren
39. `application/vnd.siren+json` internet media type to be precise

of discovered resources and using a reasoner to determine how to compose the chosen services to carry out the composition process.

While the HTTP POST method semantics have not been defined by any standards and are left open to service-specific functionality, RESTdoc restricts its semantics to be a write-once operation for the creation of a resource. Also annotations provided by RESTdoc do not allow the discovery of resources based on their functionality. Even though resource interlinking is enabled by implicit RDFS annotations, it does not fully support the HATEOAS constraint of REST.

ReLL

Resource Linking Language [Alarcon and Wilde, 2010] or ReLL is a solution for the description of RESTful Web services that focuses on their Hypermedia property (HATEOAS constraint of REST).

Descriptions in ReLL consider a RESTful service as a set of resources related to each other. Every resource in the set has its unique identifier in addition to its name, description in human readable language together with other optional properties. ReLL also explicitly indicates if a resource has multiple representations (that can be independent from the server's internal representation) in different serialization formats and supports using a URI pattern for each representation in order to avoid introducing coupling between the server and client in URI construction for requests. These representations contain in turn the links to other resource representations and constitute the means by which interlinking of the resource is established. Links in ReLL have link types that have their own name and description. These link types represent the semantics of the link in question and consequently the relation between the two resources being linked. When it comes to domain-specific annotations, ReLL allows description authors to explicitly use the vocabulary of their choice, since it does not impose any constraints on data specific ontologies.

Although ReLL does a good job capturing the interlinking of services and their resources, it does not capture explicitly the functionality aspects of a service. Functionality is implicitly indicated by link types and does not facilitate determining the behaviour of the linked resources to a given request. Also there is no support for non-functional properties and QoS aspects in ReLL descriptions.

RESTdesc

RESTdesc [Verborgh et al., 2011b] [Verborgh et al., 2012] is a resource-oriented and hyperlink-based description method for RESTful linked Web services. While it entirely relies on existing vocabularies and technologies, its novelty lies in how it combines their

concepts to describe functional service elements. It uses Notation3 [40] [Berners-Lee, 1998] or N3, a semantic Web syntax built upon RDF, to express service descriptions as well as the HTTP vocabulary in RDF [41] to semantically annotate the HTTP methods and accompanying concepts to allow clients to build HTTP requests for the described services.

From a technical perspective, RESTdesc allows service providers the freedom of choosing the vocabulary that suits the application domain of their services and adapts to it, allowing flexibility and interoperability on the data level. It also reuses the already settled stack of technologies of the semantic Web such as RDF and Notation3 and is thus compatible with existing tools such as the well established N3 reasoners. In addition to that, RESTdesc describes the precise task the service performs and leaves the semantic annotation of inputs and outputs at the charge of the description authors. On a higher level of abstraction, RESTdesc provides a scalable solution for clients to automatically discover new functionality-based interaction patterns on services instead of input/output-based ones. This is achieved by establishing preconditions and postconditions and specifying how the HTTP request should be built to perform the action and make the transition from preconditions to postconditions, rather than just semantically annotating inputs and outputs. The mentioned HTTP requests are built at runtime and are not hard-coded which enables clients to flexibly adapt to changing circumstances. It achieves that by describing relationships between resources involved in a service call and establishing the HTTP request that instantiates the functionality of those relationships.

While RESTdesc expresses descriptions of functionality in terms of preconditions and postconditions is certainly a creative and non-conventional approach, we think that the N3 powerful reasoning mechanisms on this format of description do not justify the added complexity for descriptions and client implementations. The use of simpler mechanisms for direct description via semantic annotations and semantic inferencing is, in our opinion, more advantageous and easier to integrate with Linked Data and REST practices and for description authors, who do not need a lot of background knowledge on semantic Web, to understand. Also, RESTdesc format suggests to rely on the HTTP OPTIONS method for the retrieval of descriptions, which has several drawbacks [42] most important of which being non-cacheable which conflicts with one of REST's main constraints, cacheability, especially since descriptions can be rather sizeable in the context of certain service business activities. Additionally, RESTdesc does not support any description of non-functional aspects of services, which makes selection of services hard to automate, and have to rely either on user intervention or other solutions such as service ranking and rating.

40. `https://www.w3.org/TeamSubmission/n3/`
41. `https://www.w3.org/TR/HTTP-in-RDF10/`
42. `https://www.mnot.net/blog/2012/10/29/NO_OPTIONS`

Hydra core vocabulary

The Hydra core vocabulary [43] [Lanthaler and Guetl, 2013] is the work we build upon to design our description model. It is a modular, lightweight and extensible vocabulary aimed to describe RESTful Web APIs. All of these characteristics are desired in a vocabulary for RESTful link service description :

- Modular means that descriptions are broken down into small, independent and reusable fragments, which allows partial retrieval of descriptions if needed.
- Lightweight means that description models based on Hydra are simple from a data and computation point of view, while remaining expressive.
- Extensible means that more expressive semantic annotations can be added where needed, which is exactly what we intend to do in this chapter.

Also, previous research shows that, on the Web, lightweight ontologies together with the possibility to provide custom extensions prevail against more complex models [Pedrinaci et al., 2010b] [Pedrinaci et al., 2010a]. Hydra only relies on and reuses RDFS data modelling vocabulary in order to define its own concepts, which is further proof of its simplicity, and although some of its concepts sometimes overlap with other vocabularies, the authors have chosen not to make dependencies with them because the reuse is too small to be justified.

The reason Hydra was presented as a vocabulary rather than a description model is to move from the closed-world assumption on service descriptions where description properties are defined within their own classes to the RDF's open and distributed world assumption where properties are described using vocabularies publicly available. This ensures that concept semantics are shared and reused across different application domains and service providers.

The purpose of developing the Hydra vocabulary is to simplify the development of RESTful APIs by leveraging the advantages offered by Linked Data by augmenting service descriptions to support hypermedia controls to comply with REST's HATEOAS principle, since RDF does not support hypermedia natively. Hydra aims at describing affordances offered by the Web APIs (cf. section 1.1.4) through generic concepts such as Operation and Link and uses them to augment resource representations. It describes operations at a higher level of abstraction than HTTP methods. The latter are mapped to these operations together with HTTP status codes and semantically described inputs and outputs. In addition to that it also supports semantic description of collections (similarly to W3C's LDP [44] collections) and machine readable API documentations.

43. `https://www.hydra-cg.com/spec/latest/core/`
44. `https://www.w3.org/TR/ldp/`

All these descriptions can be exchanged between the client and the server at run-time and interactions are not hard-coded into client at design time. Hence, the clients can be decoupled from the servers and adapt their execution and method calls according to changes, thus supporting loose coupling property of REST. In short words, clients can be generic.

Although Hydra supports some degree of automatic discovery with its descriptions thanks to operations and API documentation, we believe that it is not enough to cope with the large scale of the Web and is not adapted to applications such as the one illustrated by our scenario (cf. section 1.2). In fact , Hydra lacks support for cross-provider service discovery because of the way it presents the Link class that links resources to each other, which we deem not expressive enough. Also, Hydra was designed primarily to allow Web developers to describe services without much knowledge of semantic Web technologies, in order to allow a more gradual integration of service orientation with semantic Web. As semantic Web and Linked Data principles become more widely adopted and accepted, we think that Hydra descriptions should be more expressive in order to support the automation of service discovery and composition. Another drawback we observed is the lack of explicit support for non-functional properties and QoS aspects of services, which we think is vital to certain processes such as service selection.

In our work, we use Hydra as a vocabulary to annotate, with machine-readable information, descriptive elements in service descriptions. We are particularly interested in extending hydra with more expressive semantic annotations on the Link concept in order to support service discovery on the Web scale. We also extend it with support for non-functional properties and QoS attributes in order to allow service selection. We believe that these extensions would be enough to support automatic complex service manipulations such as discovery, selection and composition.

2.2.1.6 Other Related Description Efforts

Although, these are not description models in themselves, they are widely used by other description formats and are relevant to our efforts to establish an ideal description model for RESTful linked Web services.

POWDER

The Protocol for Web Description Resources [45] or POWDER aims at providing the means to describe a collection of resources belonging to the same organization, usually

45. `https://www.w3.org/TR/powder-dr/`

exposed as one or more services identified by the same URI pattern and assigned a given class, by providing meta data in the form of, inter alia, annotation properties such as `seealso` which identifies a related resource, `label` which identifies a natural language description and `comment` that identifies a comment on the resource (for example a comment on an image)

Even though POWDER provides semantics for properties to describe resources, it does not specify explicit semantics to support clients in discovering functionality within the described resources, which renders it limited in the context of functionality-focused discovery.

POWDER features the RDF property `describedby`, a property that indicates semantic linkage between two resources. The expression (formally RDF triple) :

ResourceURI1 `describedby` *ResourceURI2*

means that the resource identified by *ResourceURI2* provides a machine-readable semantic description of the resource identified by *ResourceURI1*. We use `describedby` in our solution in order to establish a semantic link between a resource and its description directly without the need to retrieve its representation through the HTTP Link header.

HTTP Vocabulary in RDF

HTTP Vocabulary in RDF [46] is a W3C effort at semantically describing the HTTP protocol element properties including HTTP methods, HTTP header elements and HTTP status codes in RDF. It gives a set of RDF classes and properties aiming to represent the HTTP specification as concepts. It also introduces new HTTP headers to facilitate the transfer of certain information regarding requests and responses.

This vocabulary aims at bridging the gap between the semantic Web practices and the practices of the REST architectural style. Many description research activities rely on this vocabulary in order to semantically annotate HTTP related description elements such as HTTP request construction patterns or header/body related elements.

2.2.2 Synthesis and Discussion

The bottom line from our bibliographic research on RESTful linked service description is that there is no widely accepted description model which would fulfil the role that WSDL plays within classic RPC Web services. Many researchers even debated whether the descriptions are needed in the case of RESTful Web services since the constraints of

46. `https://www.w3.org/TR/HTTP-in-RDF10/`

REST define and determine a lot of the behaviours and properties of RESTful services. We do not agree with this statement. We think that descriptions are very much needed in order to allow the interoperation of services from different providers. Descriptions are also needed in order to define semantics on the data and the service level. This allows clients and agents to automatically interact with and manipulate the services, discovering services that perform the needed functionality and composing multiple services for more complex tasks.

Table 2.1 is a comparative table that showcases the properties and aspects supported by the different works we presented above. This can also be considered as a different approach at categorizing them. There is several remarks we wish to discuss about this comparison.

Firstly, even though many of the description research activities have been approved by the W3C and some of them are even recommendations, this has not allowed them to gain significant adoption by service providers for the reasons we discussed above.

Secondly, we can see that the earlier propositions have used XML as a markup language to write the descriptions. Later on, XML started being abandoned in favour of more lightweight serialization formats such as JSON and JSON-LD.

Thirdly, as research community started to move towards a more lightweight approach for service descriptions, some proposals have opted to embed the descriptions directly into the services themselves. The emergence of the so called microformats has allowed an easy and lightweight description of services with a minimum required knowledge of the description mechanisms. The drawback is that there is a need to rewrite the code for the services when embedding the descriptive information, although human users will not perceive the changes directly.

Fourthly, the proposals that adopted to separate descriptions from the services and their representations allow to enforce the principle of *separation of concerns* [Dijkstra, 1982], which advocates maintaining two separate entities : the entity that provides the functionality (service or resource) and its description. This also has the advantage of having an easier process of incrementally describing services with machine-readable metadata, by adding only links to descriptions instead of the actual complete descriptions, which have to be maintained separately. The drawback is that providers have to maintain two different resources related to the same service.

Lastly, contrary to the early proposals that define and use their own semantic model, recent solutions tend to reuse the vocabularies and semantic models that are already established and approved by the normalization organizations and the research community. Some propositions try to reuse as few concepts as possible in order to keep the models

Solution	Year	Approval by W3C	Language used for descriptions	Semantic model for annotations	Level of integration with service
WSDL	2000	Submission	XML	No semantics	Separate
SAWSDL	2007	Recommendation	XML	Not specified	Separate
OWL-S	2004	Submission	XML	OWL	Separate
WSMO	2005	Submission	XML	WSML	Separate
WSDL 2.0	2007	Recommendation	XML	No semantics	Separate
WSMO-Lite	2008	Submission	Language independent	RDFS	Either
Linked USDL	2014	No	Language independent	Reuses other vocabularies	Separate
WADL	2009	Submission	XML	No semantics	Separate
hRESTS	2008	No	HTML	RDFS	Embedded (Microformat)
SA-REST	2007	Submission	HTML	RDFa	Embedded (Poshformat)
MicroWSMO	2008	No	HTML	WSMO-Lite	Embedded (Microformat)
Siren	2012	No	JSON	No semantics	Separate
RESTdoc	2013	No	JSON	Reuses other vocabularies	Embedded
ReLL	2010	No	Multiple	Reuses other vocabularies	Separate
RESTdesc	2011	No	Notation3	Reuses other vocabularies	Separate
Hydra	2013	No	JSON-LD	RDFS	Separate

Table 2.1 – Comparative table of the different description approaches

simple and the descriptions lightweight.

In the next section we present and discuss in depth our proposal for describing RESTful linked Web services.

2.3 Contribution : The Descriptor

The Web is a big network of documents resources and services. In order for a user to find the right elements that answer his requests, however complex they may be, there is a need to :

1. Automatically identify what are the basic needs that compose his request. We agree to call these basic needs tasks.
2. Automatically identify software on the Web that can answer each of the required tasks.

The answer for the first issue can be achieved by exploring the advances of language analysis as well as semantic reasoning on the concepts identified on the user's request. Both are out of the scope of our research work. We work under the assumption that this process takes as input the user's request, in the form of a text string or a data structure derived using a form or any other user interface configuration, and returns in response the list of the semantic concepts that describe the actions that need to be executed to answer the request. If we take our scenario, the request would be for example something like : "'Buy a book online"', a sting entered in a web browser or an application. The result of the process would be three semantic concepts each describing the three following operations respectively :

— Order a book.
— Choose a delivery option.
— Pay via an online transaction.

The answer for the second one, however, is not simple in the context of today's Web. The vast majority of services in the Web today do not include any machine-readable information that helps the client identify the semantics of its functionality or the semantics of the data exchanged in order to answer the user's request. The services that do contains descriptions about their functionality and behaviour feature either only syntactic descriptions or descriptions with very heterogeneous formats and differ from the rest of services on the Web. This heterogeneity complicates the design of the clients that use these services.

We aim to propose a generic and simple means to describe all RESTful services in the Web with the solution featured in the next sections. The generic aspect of our solution allows designing RESTful linked Web services in a way that promotes the construction of generic clients which can work with any service described this way in order to allow a scalable discovery and semantically interoperable distributed software on the data and service level, and ultimately allow the automatic answering of user's request. On the other hand, the simplicity aspect of our solution allows an easy way for existing services to integrate such lightweight descriptions, and future services to incorporate them in their development without requiring a lot of knowledge on the intricacies of the semantic Web technologies.

Here, we identify an important need to add descriptive metadata to the services. There are four important aspects that need to be included in the service description :

1. describe the meaning of data used by resource representations as well as inputs/outputs on the business level, by allowing the use of domain-specific ontologies chosen by the service publisher.
2. describe what is the functionality provided by each available operation.
3. describe the different relations the service has with other services on the Web.
4. describe the Quality of Service and non-functional behavioural aspects.

The first aspect is about defining the semantics of the data resources are represented in, as well as the semantics of the inputs and outputs of the different operations offered by the service. Also, since REST imposes that messages be self-descriptive and that interactions between clients and servers be stateless, these descriptions become even more important in defining the semantics of the data in the messages and requests exchanged. In the context of our scenario (presented in section 1.2), these descriptions would allow the client to know for example what is a *book title*, an *ISBN* or an *author*.

The second aspect allows the discovery process identify the services that can potentially be used in the composition process in order to answer the user's request. The discovery process crawls through the Web looking for the services that offer at least one functionality that can answer one of the tasks required by the user's request. The only information capable of identifying whether an operation can provide an answer to one of the tasks is the description of the operation itself, which is part of the description of the service as a whole. Semantically annotating each operation with a domain-specific ontology concept allows a more precise identification of the suitable services by semantically matching the concepts describing the operations of a service and the concepts describing the tasks required by the user's request. The information regarding the operations of a service

also describes the input and output required by this operations. This is helpful in the composition process where the client handles the data input and output between the different services involved in the composition. In the context of our scenario, all the client has to do in order to find a service that sells books (or that can perform the task *Order a book*) is to crawl through the Web from a given starting point until it finds a service that provides an operation that matches the concept *Order a book*.

The third aspect is another very important set of information that is required by the discovery process. As the Web contains a big network of services where each service links to another set of services in the same network, there is a need to have information about each link. This information describes why there is a link between the two services as well as the nature of this link. This is extremely vital for the discovery process, because exploring a branch in the Web instead of another can yield completely different results. The presence of the links is one of the main characteristics of an interconnected Web of services, links are what made the Web the huge information space it is, it is also one of the principles of Linked Data :

> *Include links to other URIs, so that they can discover more things.* [Bizer et al., 2009b]

Also, another important aspect in describing links is that they should be dynamic. In other words, there shouldn't be a fixed set of links, link descriptions must allow links to be added, updated and removed in order to benefit from the dynamic nature of the Web and support the scalability of service discovery. In the context of our scenario, the book selling service can link to the shipping service in order to allow the delivery of physical copies of books to the buyers. It can also add new shipping service to its links if the latter can retrieve books from the inventory of the provider and perform deliveries in the geographical areas covered. In case one of the shipping services becomes permanently unavailable for example, the corresponding link should be systematically removed.

The fourth aspect helps the selection process decide what is the most suitable service for a given operation in the case where the discovery identified multiple candidates for each task. This information combined with the information about user's preferences and behaviour help make the decision of selecting the service that will likely satisfy the user. In the context of our scenario, the client has to access the QoS descriptive data of each service it searches and compares the relevant attributes indicated by the user and then filters in the services whose QoS attributes are the closest to the users requirements.

Finally, in order to integrate with the practices of Linked Data, and consequently with the semantic Web, the service descriptions must be in RDF or a data format that can be converted to RDF and allowing the extraction of RDF triples. The semantic annota-

tions have to reuse existing vocabularies and when necessary define new domain-specific vocabularies that respect the principles of Linked Data and use their concepts in the descriptions. Also, service providers are encouraged to link the resources of their services with other related resources in order to contribute to and benefit from the interconnections of data in the Web of Data. The description in our proposal uses JSON-LD [Sporny et al., 2014], a W3C recommendation [47], which is a simple format that offers easy and compatible migration from JSON-based Web services along with the possibility of integrating Linked Data into the service description. JSON-LD is considered an RDF concrete syntax.

2.3.1 Separation of Representations and Descriptions

Most of the recent description solutions that apply for RESTful linked services we discussed in section 2.2 embed the descriptive information directly into the resource representation of RESTful services. The main drawback of this approach is the fact that service representation now contains information about both the code and data of the service and the description of its elements, operations and other related entities such as links and operations, which makes it hard for clients and applications to parse the data being described (the actual service) and the data that describes (the description) but also makes it hard for developers of clients and applications to keep track of descriptive data and complicates the building of services and the software that interacts with them.

These two sets of information address each a completely different problem. The representation of the resource is the set of data that answers the question : "what does the resource contain and what are its components ?" The description of the resource is the set of data that answers the question : "what is the nature and meaning of the resource components and data and how does the service offered and its components behave ?"

The distinction between these two sets of information can be difficult if they are not clearly separated. There is a need to separate the two sets of data in order to allow the addressing of two different problems separately, according to the *separation of concerns* principle proposed by Edsger Dijkstra [Dijkstra, 1982].

> *It is what I sometimes have called "the separation of concerns". [...] This is what I mean by "focussing one's attention upon some aspect" : it does not mean ignoring the other aspects, it is just doing justice to the fact that from this aspect's point of view, the other is irrelevant..*

The main focus of our work of describing linked Web services is to highlight the descriptive data but also give access to the information contained in the actual representation

47. `https://www.w3.org/TR/json-ld/`

of the service. This separation aids in our effort to enable automated complex interactions between Web clients and linked Web services. While the use of the actual service is enabled through the exposition of its representation, more complex service manipulations, such as : discovery, selection and composition, are made possible through the exposition of the description which is accessed by the Web client performing the aforementioned manipulations.

Our proposal also separates the description of the service behaviour (the operations allowed on the service) and the description of the service relations (the links to other related services). The available operations are easily accessed by Web clients that want to interrogate the service for a certain information (HTTP operation GET or HEAD), to create or alter the content of a certain entry in the service (HTTP operation POST or PUT) or delete a given information (HTTP operation DELETE). The service links are also made available for Web clients that wish to access other related services in case the current service does not provide an operation that help the client answer the user's request or simply look for another service that can perform the required operation better (better QoS properties, better match with the user's requirements, etc.).

2.3.2 Description Mechanisms

In order to regroup all the description aspects we talked about previously, we opted for a single data structure we call *descriptor* [Bennara et al., 2014a] [Bennara et al., 2015a] [Bennara et al., 2016b] [Bennara et al., 2016a]. Our description model details semantics of data, specific interaction possibilities with services on the Web, semantically typed links and non-functional properties of the service. We extend the representation of each of the service resources with another separate resource that contains metadata about the resource together with information about related services as well as quality of service aspects.

We explained in the previous section why we chose to separate the descriptions from the representations. In the following, we are going to explain how the descriptions are made available if we have the URI of a resource. When The URI of a resource is available to the client, it is easy to access the URI of its descriptor resource, following a generic interaction pattern. To make this interaction possible, we use the HTTP LINK header, whose semantics have been defined by the IETF [48] and W3C [49] as the means to provide the clients with more metadata about the requested resource. This is achieved by providing a link to the URI of the resource containing the metadata as well as a relationship between

48. `https://tools.ietf.org/html/rfc5988#section-5`
49. `https://www.w3.org/wiki/LinkHeader`

the requested resource and the resource linked by the URI, like this :

`Link` : <*metadataURI*> ; `rel`=*relationship*

In order to describe the relationship we chose to use the POWDER [Archer et al., 2009] `describedby` property, expressed in RDF, in order to annotate the given link with explicit machine-readable semantics that indicate to the client that the resource being linked is the descriptor of the requested resource. The list of possible `rel` values is defined by the IANA [50], and semantics of the `describedby` relation are defined using the W3C's POWDER recommendation [51]. Instead of linking to a POWDER document though, the link in question links to the data structure containing the descriptive information (the descriptor) :

`Link` : <`.../descriptor.md`> ; `rel`=`describedby`

Figure 2.1 shows the discovery of a resource and how the LINK field is accessed in the HTTP header. The client that wishes to access the description of a resource can do so by requesting an HTTP HEAD on its URI and retrieving the HTTP header. An HTTP GET would result in the same, but will prompt a response with the entire representation of the resource, and if the client is only interested in the header, this would be inefficient. After the client retrieves the header of the response, he can access the description by retrieving the value of the LINK header whose relation is `describedby`. The resource containing the description (the descriptor) can be hosted anywhere, we do not impose any condition on its URI. However, since most of the service providers would store the descriptions on the same servers as the resources they expose for efficiency, we chose to establish the following pattern for the construction of the URI of the descriptor [52] :

descriptor URI = resource URI + "/" + descriptor name + ".md"

where :

— *resource URI* is the URI of the resource being described.
— *descriptor name* is the name chosen for the descriptor resource, it could be the name of the resource or some generic name like the string "descriptor" or "metadata".
— The symbol + represents string concatenation.
— "md" stand for *meta data*, any other extension could be used, for example "jsonld", since descriptors are written in JSON-LD format, it can also be completely omitted.

The descriptor describes the resources pointing at it by giving semantically annotated information on what HTTP operation are available on the resource, together with infor-

50. Internet Assigned Numbers Authority
51. `https://www.iana.org/assignments/link-relations/link-relations.xhtml`
52. This is not a requirement, it is just a recommendation.

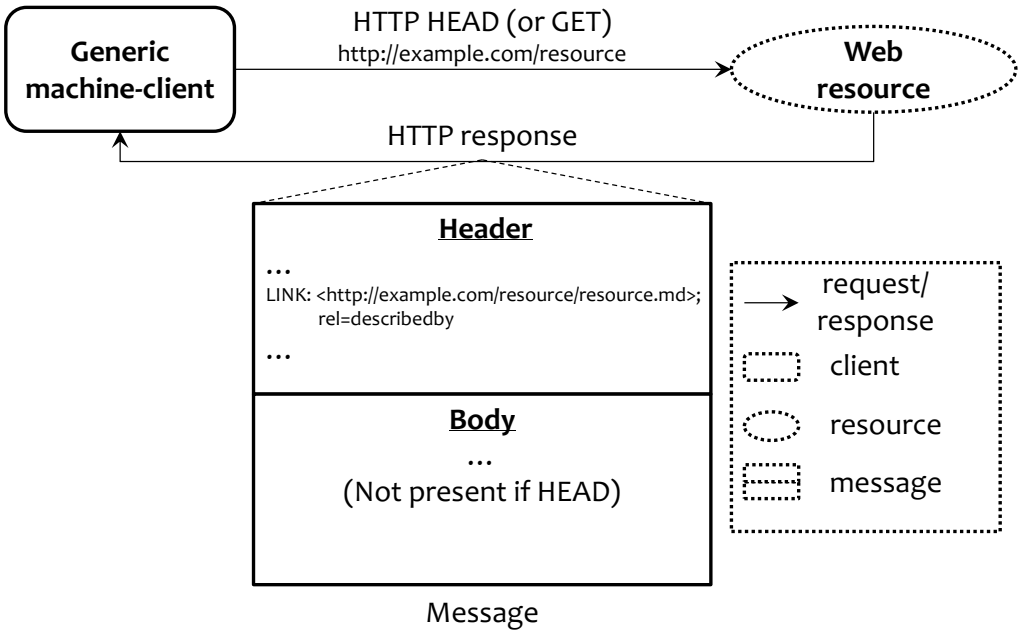

FIGURE 2.1 – Accessing a resource's descriptor URI

mation on available sub-resources, data, non-functional properties and on other related external resources, as illustrated in Figure 2.2. We call the information describing operations an *Interaction Model*, since it is the information that defines how a client interacts with a given resource.

We must underline the fact that all descriptors are resources as well according to the REST principles. Since the descriptor is also modelled as a resource, the description mechanism recursively applies. According to our proposition, every resource must have a descriptor containing metadata about it. One can get the descriptor of a descriptor via the Link field of the HTTP header received after a GET or HEAD operation on the URI of the said descriptor.

However, in this case, the Link field in the HTTP header contains a link to the universal descriptor that describes all descriptors [53]. Figure 2.3 shows how resources, descriptors and

53. Note that a GET or HEAD on the descriptor of all descriptors returns a link to itself. As the

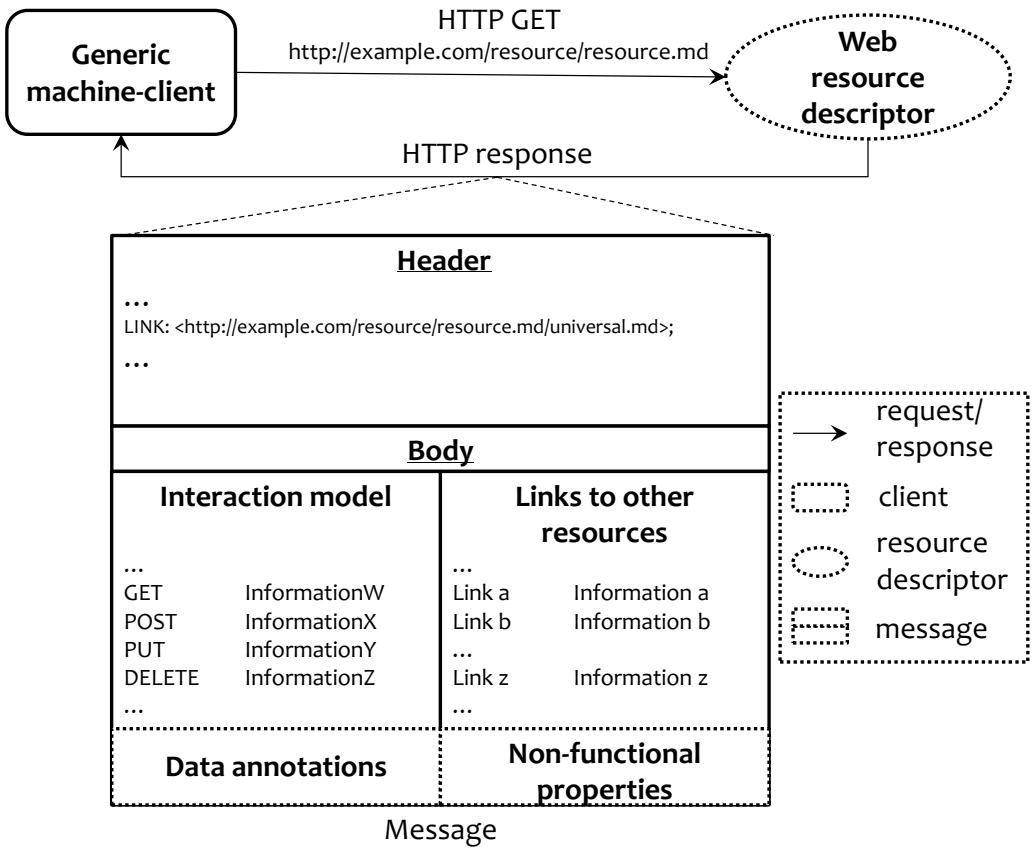

FIGURE 2.2 – Discovery of a resource descriptor

the universal descriptor are linked to each other. Links that exist between other resources are not showcased here because we are only interested on how resources describe each other.

The way clients get access to internal and external resources is homogeneous. A client accesses related resources with the required information about each of them, helping to decide which path to follow. In other words, there is no distinction between related resources that are supported by the same provider and the other resources on the Web that are supported by other providers.

The information available in the link to another resource must be helpful for the client to decide whether it should interact with the target resource or not. It must describe the semantics of the relation with the current resource and any other useful piece of information. However, the link information must not give too much details on the resource

universal descriptor is also a descriptor, it should imperatively describe itself.

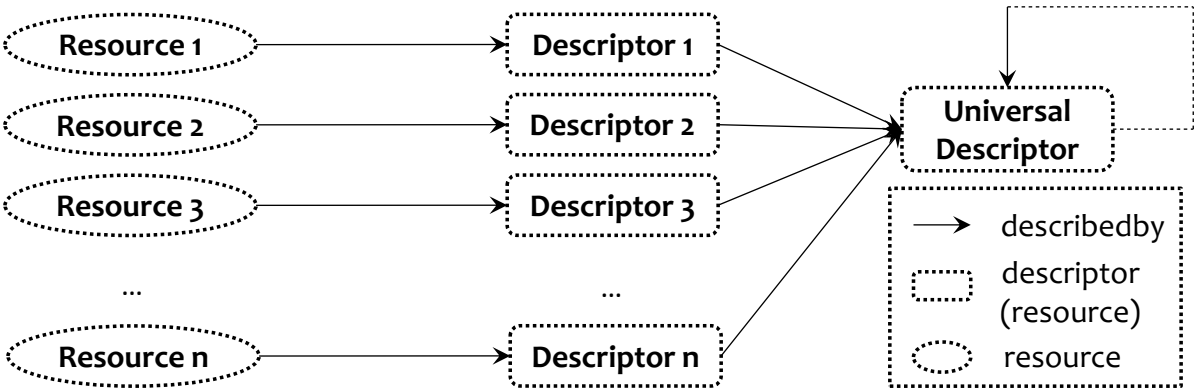

FIGURE 2.3 – Links between resources, descriptors and the universal descriptor

to avoid redundancy, data consistency and bandwidth-related problems. Details about a resource can be found in its own descriptor.

The best compromise is to only give the information that allows the client, at a given point, to decide whether to follow the path linked resource or not according to the HATEOAS REST constraint.

In the following sections we detail how different aspects of services are described in their descriptors. We emphasize on the description of operations and links since it plays a vital role in discovering functionality of services on the Web.

2.3.2.1 Describing RESTful Linked Service Operations

Describing the HTTP operations allowed on a given resource is useful for automating user request resolution, particularly when it contains information on how to correctly use the corresponding operation. Since REST defines a uniform interface for the interaction with resources in the context of the Web, that is the use of the HTTP operations to manipulate the state of resources, there is little need to define the low level semantics of the HTTP operations. The HTTP vocabulary in RDF presented in section 2.2 and the IETF HTTP protocol specification [54] already define the semantics of the GET, PUT and DELETE operations. The only operation whose semantics are not defined is the POST operation which can be used by the services for more specific functionality, and can be given the proper semantics to describe the functionality in question.

From a higher level of semantics perspective, we rely one the Hydra vocabulary [Lan-

54. `https://www.ietf.org/rfc/rfc2616.txt`

FIGURE 2.4 – Descriptor example with annotated links

thaler and Guetl, 2013] in order to describe the operations of our resources from a CRUD point of view. The information consists first of all in a description of the data-models expected and to be returned when the HTTP operation is called. The HTTP status codes that can be returned with additional details on the circumstances of each code gives details on the use of the HTTP operation. Other annotations on the operation itself, such as the CRUD basic operations (creates/reads/updates/deletes) and other advanced operations (initializes, cancels, confirms, etc.) can be useful for certain clients. Hydra defines three types of operations *CreateResourceOperation*, *ReplaceResourceOperation* and *DeleteResourceOperation*. Annotations on the data manipulated by the operation makes its semantics explicit in order to allow a better exploitation by the client.

2.3.2.2 Describing RESTful Linked Service Links

Our descriptor-based solution allows generic clients to crawl from one resource to another in order to select interesting resources to answer the user's query. However, due to the huge number of resources on the Web, there is a need to help the discovery algorithm to only select the most interesting resources. We optimize our descriptions on links in order to work with the BFS (Breadth First Search) algorithm, as this is the best general crawling algorithm for the Web structure (We talk about this in detail in Chapter 3). We introduce semantic annotations on descriptor links. The semantic annotations on links will guide the algorithm by excluding irrelevant links to the current application and taking into account the links that potentially point to resources that offer functionality that can answer part of the user's request. Fig 2.4 gives an example, relating to our scenario, of the semantic annotation of descriptor links. The ontology namespaces used for operations are for illustration purpose only, in a real world application the semantics would be given by domain-specific vocabularies, since our description approach does not impose any vocabulary to annotate operations.

As for describing links, the semantics are inspired from existing work [Maamar et al., 2011b] to define the properties that link resources to each other. We define that :

— Two resources are `similar` if they provide functionally substitutable services, sometimes varying in terms of non-functional properties.

— Two resources are `complementary` if they can be combined in the same process to answer user's needs, for example : a flight booking service and a hotel booking service inthe context of a trip.
— Two resources are `incompatible` if they cannot be involved together in the same process because of a given reason, for example : the eBay online seller could decide not to work with the UPS delivery company.

Since, our description model is modular and extensible, other types of semantic annotation could be used to annotate links between resources, we chose this one in order to support the discovery solution we propose in chapter 3.

2.3.2.3 Describing Service Data and Non-Functional Properties

In order for clients to be able to deduce the meaning of the data exchanged with the service, annotating the data elements and the operation inputs/outputs is necessary. Our description model does not impose any specific vocabularies for the description of data. Description authors and service providers are free to use the domain vocabulary of their choice, or the one most adequate to the business domain of the service.

As for non-functional properties, there are some research activities in order to define common ontologies to describe the semantics of the properties. For the purposes of illustration of our research, we chose to use a very simple ontology that defines semantics of a limited number of chosen properties, but our model does not specify that service providers have to use it in service descriptions. The minimal QoS model is defined and discussed in chapter 3.

There are also some descriptions that need to be implemented in the descriptor as well, but they are irrelevant in the context of our research, as our main focus is to allow the automatic answer of a user's request involving Web services. Such descriptions include information about provider, human-readable description, provenance data, service area, language, location, statistics, etc.

In order to support the traceability of discovery and composition processes in our solution we can rely on the fact that resources can keep track of the interactions they have been involved in, and publish these traces, after processing, via their descriptor. Several possibilities exist to record resource interaction, such as storing the addresses of incoming requests, these addresses may lead to other related resources which the provider may add later by analysing this data. Another solution is to extract well-known mashups from Web sites such as ProgrammableWeb and other similar data-bases and add links to services that are used the most in conjunction with their own. How to build and maintain

such a list is out of the scope of our research work.

2.3.3 Guiding Discovery, Selection and Composition

We talked in previous sections about the important role the descriptions of a service in determining how it is discovered and how it helps clients decide which service to call upon for a certain task. In the following, we are going to explain how different description parts contribute to discovering, selecting and composing services in order to answer the user's request.

Our solution relies on a generic client that interacts with the different services through their respective APIs on one side and with the user through its GUI on the other. The client needs to be able to automate the process of discovering, selecting and composing the functionality of a number of services in order to answer the user's query.

The main information that guides the discovery is the description of functionality (operations/inputs/outputs) and links. The description of functionality helps the client answer the questions :

— Does this service provide a functionality needed to perform a task required by the user's request ?
— What are the requirements on inputs ?
— What are the outputs/results to expect after I call upon the current service ?

For example, in our scenario it helps the client decide if the service currently being discovered offers the possibility of buying a book online, offers shipping of books bought from an online store or offers the possibility to make a cashless payment to an online service.

The description of links helps the client answer the question : "Where can I find services that provide functionality similar or complementary to the current service ?" or more generally "Where can I find more services related to the current service ?". This is what allows the client to explore the huge graph of services on the Web. The efficiency the of discovery algorithm depends on this information. The semantic annotations on the links are what determines the order they are traversed and consequently the quality and relevance of the results of discovery.

As for selection, the main information that drives the process is non-functional descriptions about the services being discovered. It helps the client make the decisions regarding whether or not the context of execution of the service is relevant to the user's context and also whether its quality of service attributes match the user's expectations. For example, in the context of our scenario the information about the service's operating geographical

area, the book languages, the currency used, etc. is exploited by the selection algorithm and compared to the user's profile in order to compute the eligibility of the services discovered to be used to perform the tasks required by the user's request. Also, the clients expectations in terms of pricing, availability, shipping delays, etc. are taken into account by the selection algorithm and compared to the discovered services descriptions to filter out the ones that are not up to the requirements.

When it comes to composition, after the discovery and selection processes have determined a service for each task, the composition algorithm starts orchestrating the different calls to those services and handles the data flow between different operation inputs and outputs. The descriptions of the semantics of the data resource representations as well as operation inputs and outputs contain the information necessary to carry out this process. In the example of our scenario, the client can determine with the help of these description elements if the shipping service can deliver the user's order on the book selling service, knowing the order's weight and dimensions for example. Also, it can determine if the payment service can retrieve money from the user's bank account and transfer it to the vendors'.

We dedicate chapter 3 to a more detailed discussion about discovery and selection and chapter 4 to a more detailed discussion about composition of services described according to the mechanisms we detailed earlier in this chapter.

2.3.4 Applying the Description Mechanism to the Motivating Scenario

The applicability of our description proposal and in particular that of the discovery process is demonstrated with the help of our scenario presented in section 1.2. We implement a generic client that takes advantage of the given descriptors in order to decide, at every moment and according to the user's needs, what resource links it should follow. While interacting with the different resources, the generic client handles the application state until it fulfils the user's goal. The transitions of the application state are dictated by the client on the basis of the representation and the description of the resource being accessed as well as previous interactions with other resources. Figure 2.5 shows how our generic client discovers the different resources needed in our scenario. The entry point, which is the URI of the homepage of the book selling service, is given to the client that follows the links in that resource searching for services and resources that can fulfil the tasks dictated by processing the user's request.

We present here the scenario and how it operates according to the elements of our

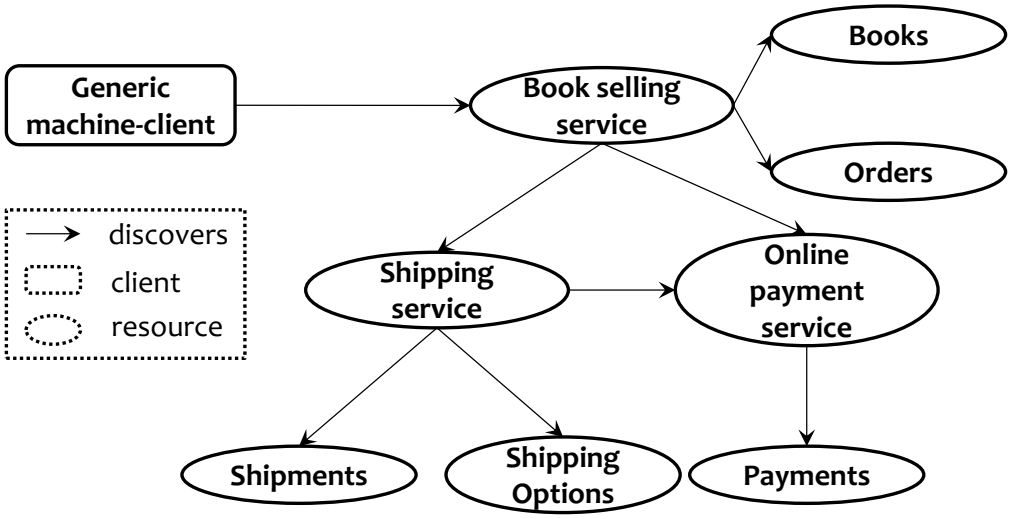

FIGURE 2.5 – Service Discovery in our Scenario

contribution. Typically, a HEAD (or GET) HTTP operation on any resource should return the URI of the resource description (`http://example.com/resource/resource.md`, where md stands for metadata) in the LINK field of the HTTP header, as described in Fig. 2.1.

A GET operation on the URI of the link returns the descriptor, which is also a resource. The descriptor contains links and all the necessary information about the sub-resources of the resource it describes (for example containers), and about external related resources. As we talk about a unified method for semi-automatic discovery, the descriptions of the internal sub-resources and external ones are homogeneous and are processed the same way on the client side.

The description allows the generic client to explore both internal and external interaction possibilities according to the information given on each resource. The client knows what is the best HTTP operation to apply to the resource according to its needs and the

information available in the obtained description.

In the following, we present the resources modelled in our scenario and give their corresponding URIs. The particular resources are stored in their respective repositories (books, shipping options, shipments, payments) and have the following URI patterns :

— Particular book : /bookselling/books/book123
— Particular orders : /bookselling/orders/order123
— Particular shipping options : /shipping/options/option123
— Particular shipments : /shipping/shipments/shipment123
— Particular payments : /payment/payments/payment123

The interactions between the user/client and the different services are detailed in the following. The user wants to browse the different books offered by the book selling service. The book selling service URI is either chosen by the user at first as the entry point for the client, or discovered by the client using the high level request given to it by the user, using the entry point given. The book selling service allows the users to create a virtual shopping cart and add books they browse to it. After the user is done picking up books, the shopping cart is submitted and an order is created. The user has to chose a shipping option for the delivery of his books. After that, he has to pay both the book vendor and the shipping service provider via a payment service, where he has to enter his credit card details (the client can also add this automatically or the payment service could offer to auto-complete the details if the user gave permission to remember his details in a previous purchase), and submit the payment that validates his purchase and terminates the book buying process. From the point of view of the machine client, the steps necessary to carry out such an interaction is detailed in the following.

Firstly, the user wants to browse book descriptions [55] (GET on `books repository` resource) to select one or several `book` resources, whose detailed descriptions are then stored client-side. The problem is that the client at first does not know the URI of the books repository. It discovers this URI using the descriptor of the book selling resource. The link to this descriptor is found in the HTTP response to a HEAD (or GET) operation on the entry point, we suppose known by the client, represented by the URI of the book selling service /bookselling/. More precisely, in the very beginning of the execution flow, the client executes a HEAD operation on the entry point URI. It gets the URI that corresponds to the `describedby` property attached to the Link Header field of the HTTP response, which is /bookselling/bookselling.md in our scenario. Then, it performs a GET operation on this link to get the descriptor of the book selling service. In this descriptor, the client finds multiple links to other related resources. It chooses the one pointing at the

55. We refer here to the actual book description with its URI, author, title, price, and so on.

book repository using the semantics provided with the link and according to the user's needs. Another question that our contribution answers is : how does the client know what HTTP operation to execute in order to obtain book descriptions. The answer is : the client accesses the metadata on allowed HTTP operations and decides which one responds to the objective of getting a book description, in this case it is the GET operation.

Secondly, there is a need to create an `order` (POST) on the `orders repository` resource. The latter is referenced via the metadata contained in "bookseller.md" retrieved from the book selling also exactly the same as the `books repository` resource. The HTTP operation to execute is retrieved exactly as explained above. Once the `order` resource created, it remains in the status "Unpaid". It can be updated (PUT) to "Paid" once the payment is confirmed. The client is not responsible for verifying whether or not the payment is successful. This is achieved via automatic, inter-service interactions that are out of the scope of our scenario [56]. Then, the link to the shipment service is extracted with the help of the "bookseller.md" resource that includes external resources.

Thirdly, a shipment option [57] is selected (GET) and a new `shipment` is created (POST) and set to the status "Unpaid", waiting for an update on the payment confirmation (PUT). The `shipment` created is associated with the created `order` and the information necessary for the delivery is also stored with it. The link to the "payment" service is also retrieved by accessing the "bookseller.md" or "shipping.md" resource.

Finally, the `payment` resource is created (POST) using the user's credit card or bank account details and is submitted to the payment service, who then performs the necessary bank transactions, and if everything is okay the payment is confirmed as effective and the `order` and `shipment` resources are marked as paid.

2.4 Implementation and Technical Design Choices

In this section, we give the specification of our proposed solution for the description of RESTful linked Services. First, we start by introducing JSON-LD, the data format we use to represent our descriptions. We debate why it is suited to describe RESTful Linked services, present its features, discuss its backwards compatibility with JSON and enumerate its goals. The descriptor uses JSON-LD as a data format to present descriptions in a machine-readable format to the machine clients and agents, in order to automate their

56. For the sake of brevity, we omit security concerns, but such an exchange may involve key sharing and secured protocols such as Oauth (`http://oauth.net/`).

57. By shipment option, we mean the different modes of delivery this kind of services offer, such as delivery under 24 hours, or one week, etc.

service manipulation tasks. After that, we specify the structure of the descriptor which is divided into four different parts : operation descriptions, link descriptions, non-functional descriptions and data/service descriptions. Lastly we discuss and illustrate the structure of the descriptor and the impact of the design choices we made.

2.4.1 JSON-LD

JSON-LD [58] [Sporny et al., 2014] is a lightweight Linked Data format. It is easy for humans to read and write. It is based on the already successful JSON format and provides a way to help JSON data interoperate at Web-scale. JSON-LD is an ideal data format for REST-based Web services because it can be easily parsed into ready-to-use JavaScript objects. As of now, JSON-LD is a W3C recommendation [59] and is still being developed [60].

The syntax of JSON-LD is designed specifically to integrate into deployed systems that already use JSON, and provides a smooth upgrade path from JSON to JSON-LD. Its design allows existing JSON to be interpreted as Linked Data with minimal changes. JSON-LD is primarily intended to be a way to use Linked Data in Web-based programming environments, to build interoperable Web services, and to store Linked Data in JSON-based storage engines. Since JSON-LD is 100% compatible with JSON, the large number of JSON parsers and libraries available today can be reused. In addition to all the features JSON provides, JSON-LD introduces :

— a universal identifier mechanism for JSON objects via the use of IRIs [61],
— a way to disambiguate keys shared among different JSON documents by mapping them to IRIs via a context,
— a mechanism in which a value in a JSON object may refer to a JSON object on a different site on the Web,
— the ability to annotate strings with their language,
— a way to associate data types with values such as dates and times,
— and a facility to express one or more directed graphs, such as a social network, in a single document.

JSON-LD is designed to be usable directly as JSON, with no knowledge of RDF. It is also designed to be usable as RDF, if desired, for use with other Linked Data technologies like SPARQL. Developers who require any of the facilities listed above or need to serialize an RDF Graph or RDF Dataset in a JSON-based syntax will find JSON-LD of interest.

58. `https://json-ld.org`
59. `https://www.w3.org/TR/2014/REC-json-ld-20140116/`
60. `https://www.w3.org/2018/jsonld-cg-reports/json-ld/`
61. Internationalized Resource Identifiers `https://tools.ietf.org/html/rfc3987`

People intending to use JSON-LD with RDF tools will find it can be used as another RDF syntax, like Turtle. The syntax is designed to not disturb already deployed systems running on JSON, but provide a smooth upgrade path from JSON to JSON-LD. Since the shape of such data varies wildly, JSON-LD features mechanisms to reshape documents into a deterministic structure which simplifies their processing.

JSON-LD is designed to satisfy the following goals :

— Simplicity : No extra processors or software libraries are necessary to use JSON-LD in its most basic form. The language provides developers with a very easy learning curve. Developers only need to know JSON and two keywords (@context and @id) to use the basic functionality in JSON-LD.
— Compatibility : A JSON-LD document is always a valid JSON document. This ensures that all of the standard JSON libraries work seamlessly with JSON-LD documents.
— Expressiveness : The syntax serializes directed graphs. This ensures that almost every real world data model can be expressed.
— Terseness : The JSON-LD syntax is very terse and human readable, requiring as little effort as possible from the developer.
— Usable as RDF : JSON-LD is usable by developers as idiomatic JSON, with no need to understand RDF. JSON-LD is also usable as RDF, so people intending to use JSON-LD with RDF tools will find it can be used like any other RDF syntax.
— Zero Edits, most of the time : JSON-LD ensures a smooth and simple transition from existing JSON-based systems. In many cases, zero edits to the JSON document and the addition of one line to the HTTP response should suffice. This allows organizations that have already deployed large JSON-based infrastructure to use JSON-LD's features in a way that is not disruptive to their day-to-day operations and is transparent to their current customers.

2.4.2 JSON-LD and RESTful Linked Services

As the Web shifts towards the Linked Data, using JSON-LD as a data format yields more benefits than the classically used XML-based formats in Web applications, since XML-based formats are much more difficult to parse and also because JSON-based formats can be parsed into ready-to-use JavaScript objects. In addition to that, using JSON-LD as a data format in the Web of data allows for a full exploitation of the Linked Data and Web of data potential, which remained, until the last few years, largely unexploited [Pedrinaci et al., 2010a] [Pedrinaci et al., 2010b]. Since JSON-LD is a concrete RDF syntax, this means that descriptors are formally RDF graphs. JSON-LD supports Linked

Data principles and allows the use of vocabularies and name spaces to describe properties with semantic annotations. JSON-LD is a simple format that offers easy and compatible migration from JSON-based Web services along with the possibility of integrating Linked Data into the service descriptions.

2.4.3 Hydra core vocabulary

Hydra [Lanthaler and Guetl, 2013] is a modular, lightweight and extensible vocabulary aimed to describe RESTful Web APIs. Modular means that descriptions are broken down into small, independent and reusable fragments, which allows partial retrieval of descriptions if needed. Lightweight means that description models based on Hydra are simple from a data and computation point of view, while remaining expressive. Extensible means that more expressive semantic annotations can be added where needed. As we discussed in section 2.2, lightweight solutions for description combined with the possibility to provide custom extensions are superior to more complex models, in the general context of the Web. The purpose of developing the Hydra vocabulary is to simplify the development of RESTful APIs by leveraging the advantages offered by Linked Data by augmenting service descriptions to support hypermedia controls to comply with REST's HATEOAS principle, since RDF does not support hypermedia natively. Hydra aims at describing generic concepts such as Operation and Link and uses them to augment resource representations.

Our description model is built upon the concepts introduced by the Hydra core vocabulary. We extend the basic descriptive structure supported by Hydra in order to allow automatic service discovery, selection and composition. Although Hydra's operations and links support some degree of automation in service discovery, there is not enough semantic annotations given to these elements to support a fully automatic service description in a Web of resources. To this end, we dedicate more semantic annotations over operations as well as links in order to support the machine client responsible for carrying out the discovery process (cf. section 2.3 for more conceptual details). Also, we extend the Hydra model with a QoS model to support the description of non-functional properties and allow a fully automated selection process (cf. section 2.3). Combining the results of the discovery and selection together with the data/service descriptions, the composition process can be automatically carried out with a full knowledge of the semantics of the functionality as well as the inputs/output of the different services involved in the composition.

2.4.4 Technical context of the specification

In order to demonstrate the feasibility of our proposals, we opted implement the services and resources involved in the scenario we introduced in chapter 1. The method of choice we opted for to implement the scenario services is *Java Servlets* using the *Jersey framework*, which is based on the *JAX RS* API. Our IDE [62] of choice to organize and edit the difference projects for each Java Servlet is *Eclipse*. In order to accommodate and expose the Java Servlets representing services on the Web, we use *Apache Tomcat* as an application server (Java Servlet container). We use *Gson* module in order to ensure the serialization/deserialization of POJOs [63] into/from JSON(-LD). We also use *JavaScript* in our generic client as a client-side scripting language.

2.4.4.1 Java Servlet

In a general context, a Java Servlet is a software component written in Java acting as a server in a client-server architecture. It is capable of handling requests (most commonly HTTP requests on the Web) and constructing a response to those requests. In other words, Java Servlets are used to construct server-side Web applications in a request-response interaction model, using Java.

In the context of our work, Java Servlets are used to construct RESTful services that provide functionality on the Web. From the developer's point of view, the servlet handles all the HTTP requests targeted at the service they represent and at the sub-resources contained within it. From the client's point of view, the service provided by the servlet is a set of individual resources each capable of responding to a determined set of HTTP requests.

In order to host and expose the functionality provided by the Java Servlet, we need to use a servlet container in order to handle the incoming request on a lower and more technical level.

2.4.4.2 Jersey Framework

The Jersey [64] is an open source RESTful Web Services framework for developing RESTful Web Services in Java. Jersey's main goal is to simplify the development of RESTful Web services and their clients in Java. It aims to abstract away the low-level details of the

62. Integrated Development Environment
63. Plain Old Java Object (or simply Java Object)
64. `https://jersey.java.net/`

client-server communication and to seamlessly support exposing data in a variety of representation media types. Jersey also exposes numerous extension APIs so that developers may extend to best suit their needs.

In the context of our work, we use Jersey to implement and define the APIs of the different scenario services, in the form of Java Servlets. It allows the handling of HTTP requests such as GET, POST, PUT and DELETE as well as the representation of other HTTP related concepts such as a URI path, media types, HTTP request body and header, HTTP response body and header, etc.

2.4.4.3 Gson module

Gson [65] is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of. There are a few open-source projects that can convert Java objects to JSON. However, most of them require that you place Java annotations in your classes ; something that you can not do if you do not have access to the source-code. Most also do not fully support the use of Java Generics. Gson aims to fulfill the following goals :

— Provide simple toJson() and fromJson() methods to convert Java objects to JSON and vice-versa.
— Allow pre-existing unmodifiable objects to be converted to and from JSON.
— Extensive support of Java Generics.
— Allow custom representations for objects.
— Support arbitrarily complex objects (with deep inheritance hierarchies and extensive use of generic types).

In the context of our work, we simply use Gson in order to serialize/deserialize Java Objects in the Java Servlets into JSON-LD that is exposed on the Web as descriptions and representations of the resources constituting the different scenario services.

2.4.4.4 Apache Tomcat

In the context of our work, Tomcat [66] is an open source Java Servlet container that can host, expose and run server-side Java code provided by the said Java Servlets. In other words, a Tomcat server represents the server-side application that receives and handles HTTP requests from clients and executes the code provided by Java Servlets in order to

65. `https://code.google.com/p/google-gson/`
66. `http://tomcat.apache.org/`

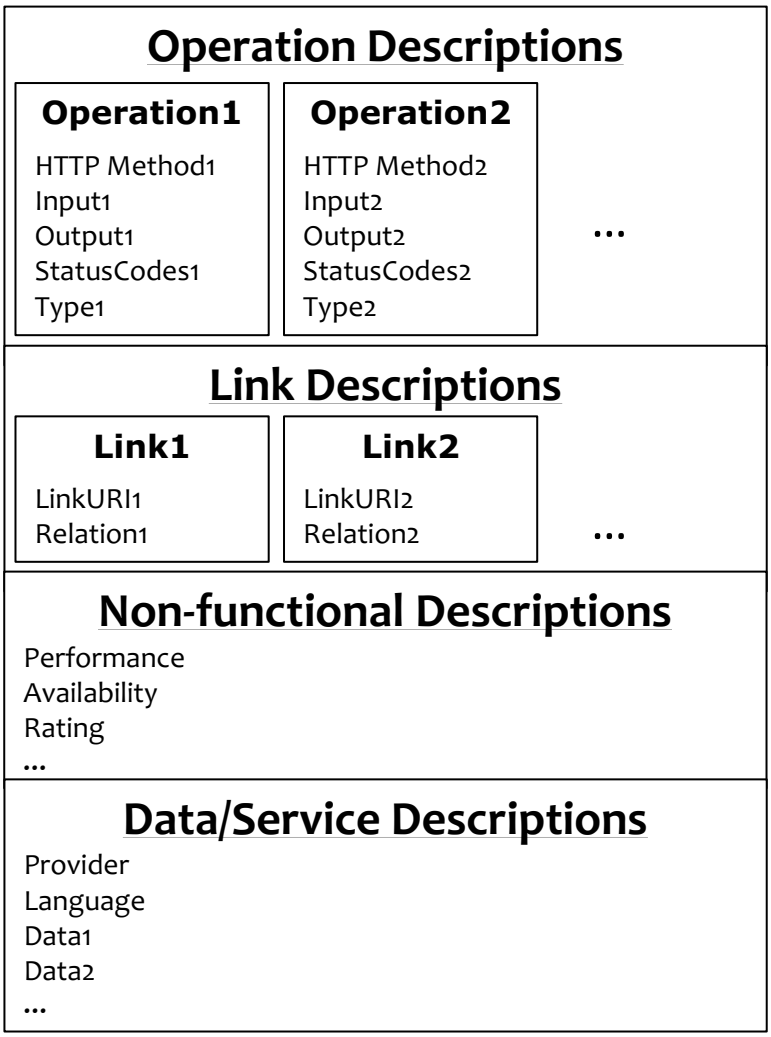

FIGURE 2.6 – Structure of the descriptor from a conceptual point of view

construct HTTP responses to these requests and return them to the requesting clients.

2.4.5 Specification of the descriptions

From a high level of abstraction, the descriptor of a resource is a data structure in JSON-LD comprised of four different description aspects. Figure 2.6 illustrates the structure of a descriptor from this point of view.

2.4.5.1 Specification of the operation descriptions

This section presents how we describe the functional aspects of the resource being described, by semantically annotating its operations and its inputs/outputs. This is the

descriptive information that allows users and clients to identify what a resource does in the context of its service.

On a more technical level, the operation descriptions contain multiple descriptions for each operation. Each operation description contains the properties defined by hydra namely : the HTTP method, the inputs/outputs and the expected statues codes. In addition to these properties, we introduce the *Type* attribute that allows to annotate the operation in question with a semantic concept from a domain ontology matching the functionality provided by the said operation. This property is the main guiding element for the automation of the discovery process. In other words, the discovery algorithm relies on the *Type* property in order to determine whether or not the functionality provided by the described operation matches with the discovery criteria.

Figure 2.7 illustrates the structure of a single operation description. If the resource being described supports more operations, the descriptions for these operations can be found alongside it. Note that we omit the definition of certain properties such as the JSON-LD *@context* and *@id* for the sake of brevity.

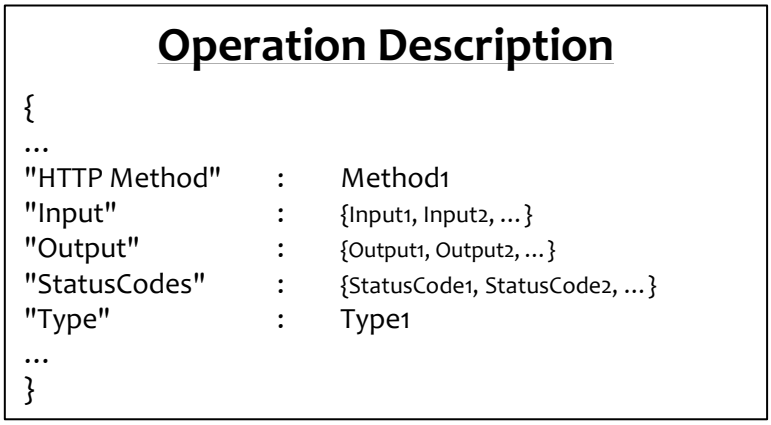

FIGURE 2.7 – Structure of a single operation description

2.4.5.2 Specification of the links descriptions

This section details the way we describe the links a resource contains, that point to related resources. The links in question either point to internal resources related to the functioning of the current resource or external resources that relate somehow to the service that the current resource is part of.

On a more technical level, the link descriptions, like their operation counterparts, include multiple descriptions for each link available to other resources. Each link description

contains two properties : *LinkURI* and *Relation*. *LinkURI* contains the URI of the target resource, and can be used by the client to access the resource and its descriptor. *Relation* describes the nature of the relationship (the *directed* arc, if we talk in a graph context) that binds the current resource with the target resource. As discussed in section 2.3 and chapter 3, we chose to use values to express the compatibility between the services being linked (Similar, Complementary, Incompatible). This choice was made in order to support the *Relation* property and illustrate the feasibility of a fully automatic service discovery based on semantic annotations on operations and links. Our description model remains flexible and extensible however, meaning another set of values can be used instead as long as it provides a solid base upon which the discovery algorithm can discover new services that perform the functionality needed by the client. Our link descriptions differ from the link descriptions provided by Hydra because we think that Hydra links are lacking the necessary expressivity to support a fully automated service discovery process.

Figure 2.8 illustrates the structure of a single link description. More link descriptions can be found alongside if the resource has other links.

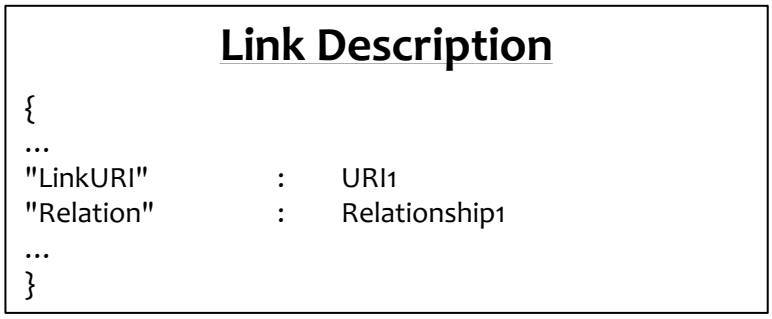

FIGURE 2.8 – Structure of a single link description

2.4.5.3 Specification of the non-functional descriptions

In this section, we present the details of the non functional aspects descriptions and their semantic annotations. As we stated in section 2.3 and chapter 3, we do not use a complete Quality of Service model, but only a minimal one to demonstrate how the selection process filters out the unwanted services located by the discovery process, on the fly.

On a more technical level, the non-functional descriptions contain quality of service properties. As discussed in chapter 3, the QoS model we went with is minimal and only serves an illustration purpose. Again, the flexibility and extensibility of our proposal allow the use of a more elaborated QoS model that better reflects the simple and more intricate

business-level quality properties and requirements that have to be up to the level required by the user. What is important is that the properties represented in this part of the description can be compared with the user requirements in order to filter out undesired services that do not match with user expectations.

Figure 2.9 illustrates the non-functional descriptions using the minimal QoS model presented in chapter 3.

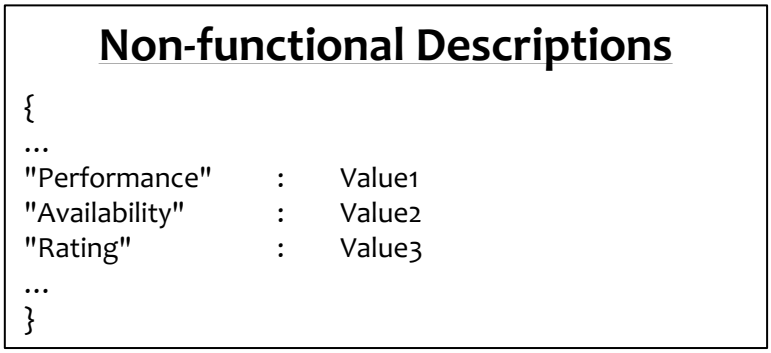

FIGURE 2.9 – Structure of non-functional descriptions

2.4.5.4 Specification of the data and service descriptions

We detail, in this section, the way data and service element descriptions are presented in the context of their resource descriptor. These descriptions include the semantic annotations given to different data and service related properties contained in the resource representation. These descriptions are important because they allow the client that invokes the service to understand the meaning of the data the resource provides. They also allow the client to access important service related information such as provenance and service statistics.

Figure 2.10 illustrates the general structure of data and service descriptions within their resource descriptor.

2.4.6 Summary

We presented in this section the specification for our proposed contribution. We demonstrated how service descriptions are separated and represented in JSON-LD. JSON-LD descriptions ensure backwards compatibility with systems that already work with JSON, whose popularity is increasing with the rise of RESTful APIs as the dominant service-oriented implementation. JSON-LD is also a developer-friendly format due to its

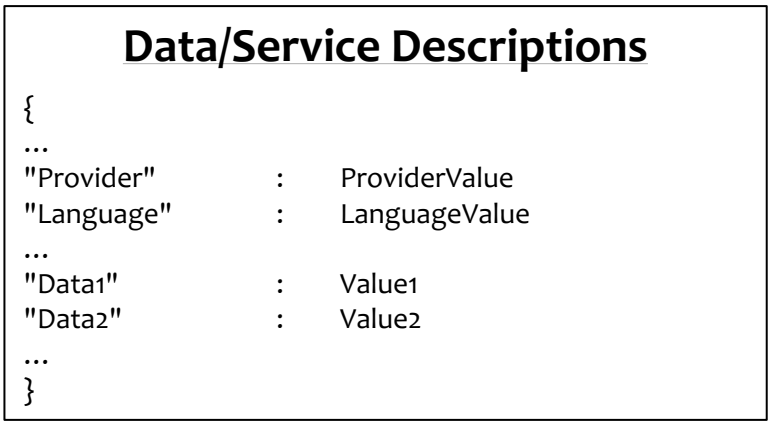

FIGURE 2.10 – Structure of data and service descriptions

inherent simplicity and terseness in addition to supporting Linked Data and RDF for more expressive and semantically annotated descriptions. We also discussed how we reuse the Hydra core vocabulary in order to support and extend the notions of *Operation* and *Link*. We explained how we extend the Hydra concepts in our descriptors to support a fully automated discovery, selection and composition of RESTful Linked Services. We detailed and illustrated the different descriptive information contained within the descriptor and explained the role each description plays and we explained how the flexibility and extensibility of our solution allows for more detailed and expressive descriptions when needed.

2.5 Conclusion

In this chapter, we studied the different research activities aimed at describing RESTful linked Web services. We highlighted the difficulties the research community had in describing RESTful services and integrating the descriptions with the emerging Linked Data publication practices. We revealed that the classic Web service description models are obsolete in the context of a RESTful architecture and that more lightweight, modular and extensible solutions for descriptions are superior. However, even the later solutions showed some limitations, mainly due the fact that they do not adhere to the full set of constraints advocated by the REST architectural style, or they focus on either over-describing links or operations and ignoring other important descriptive information.

We sketched a description model that allows the integration of the RESTful service orientation with Linked Data practices and the semantic Web and we proposed a novel approach to concretely describe RESTful linked Web services, we dubbed *Descriptors*. Our

proposal relies on the information needed to answer a complex user request in order to identify exactly what information needs to be present in the description of a Web service. We identified that interaction models and interlinking of resources in services are the most important aspects that drive client discovery of new services to answer the user's request.

Thanks to this description mechanism the client responsible for answering the user's request can carry out the process by exploring descriptions of discovered services and decides if it is useful or not and if it is how is the client going to use it thanks to the semantic annotations. With the help of the scenario, we demonstrated the feasibility of our contribution and how we overcame the challenges.

Our proposal consists in attaching a descriptor resource to each Web service. A link to the descriptor is obtained from the LINK header of a HTTP HEAD (or GET) response from the resource URI. Ultimately, all resources including descriptors need to be described, which led us to conceive a universal descriptor describing all descriptors of resources including itself. A resource descriptor contains useful information about the service, its operations, its behaviour and the related services.

A descriptor contains useful information about the resource and how it interacts with clients and other resources. It contains links and information about how other resources relate to it. By following these links and making use of the available information about the available operations, the client handles the discovery, selection and composition processes in order to fulfil the user's objectives.

The generic client is responsible for carrying the process of answering the user's request by accessing resource descriptions, discovering new resources, selecting the relevant ones and carrying our the data exchanges from one resource to another to finally show the results to the user.

From another perspective, our resource/descriptor model implicates separating the representation and the description aspects of a service. This separation several implications. The most notable one is that accessing the descriptive information requires two HTTP operations : the first one on the service URI itself and the second one on the resource's descriptor URI. The drawback is that extensive accessing to different services generates additional computing time for the Web client because the first time it performs an additional operation in order to gain access to the descriptor contents. This drawback, in our view, is largely compensated with the fact that the description process of already existing resources can be performed incrementally with minimal effort and knowledge of the intricacies of the semantic Web technologies.

The design of new services is also simplified by the fact that separation of representations and descriptions follows the guidelines of the *separation of concerns* advocated

by Dijkstra. Also, additional access to the same resource now require less actions (and thus less computing time) thanks to the cashing mechanisms which are guaranteed by the principles of REST. This means that accessing the same resource in order to execute a repetitive user's request, for example, will have a positive impact on the response time.

As a future work, a response time study could be made comparing different setups where resource representations and descriptions are either separated (like in our proposal) or merged (while keeping the descriptor structure, or embedding descriptions directly into service elements). Different scenarios can be taken into consideration where resources are either accessed repetitively for the same (or similar) task or where resources are diverse and rarely accessed twice.

In the remainder of this thesis, we show how we use this descriptive information in order to facilitate the discovery of services and help make decisions in the selection process about what services to be selected for the different tasks. We also show how the description of data and inputs/outputs of operations help carrying out the composition process.

The next step in our work is to carry out the discovery and selection processes given the information present in descriptors.

Chapter 3

RESTful Linked Service Discovery and Selection

Contents

Automating discovery and composition of RESTful services with the help of semantic Web technologies is a key challenge to exploit the full potential of today's Web. Recently, resource oriented computing has changed the way Web applications are designed. Because of the increasing number of RESTful APIs, centralized repositories are no longer a viable option for discovery. As a consequence, a decentralized approach is needed in order to enable value-added applications.

In this chapter, we show how semantic annotations on resource descriptions can drive scalable discovery algorithms on the Web. We propose a semantically-enabled variant of the BFS discovery algorithm that aims at minimizing the number of links explored while maximizing result diversity. Our algorithm calculates semantic distances between resource descriptions and user request concepts to rank explored resources accordingly. We demonstrate the applicability of our solution with the help of the motivating scenario. At the same time, since the graph of services the discovery algorithm traverses is not fully visible at the start, we propose a client-side QoS-based selection algorithm that can be executed along with the discovery process. Our solution provides different setups based on the skyline approach to select resources and maintain acceptable time performance.

3.1 Introduction

The subject of service discovery has seen a fair amount of research activity since the introduction of Web services. With the introduction of the service orientation to the Web at large, there has been a lot of work in order to propose solutions that allow a scalable way of discovering functionality on the Web. The introduction and emergence of the Linked Data and the semantic Web technologies has allowed the interlinking of huge amounts of semantically annotated data on the Web, which prompted the need for software that provides functionality to process this huge amount of data. The advent and spread of RESTful APIs has allowed to provide such functionality on an architecture that is scalable, and Linked Data allowed the semantic interoperability on the data level. On the service level however, there has been a lot of research in order to allow the interoperation of heterogeneous services in the form of description models to describe syntactically and semantically the functionality of services so that clients and automated agents could discover functionality that processes the huge amount of linked data on the Web.

The discovery of functionality on the Web in the form of service discovery has been hampered by the fact that no solution to describe Web services has gained enough traction with the service providers to overcome the heterogeneity on the service level. The

majority of RESTful services are only described in human-readable labels and in order to use their functionality there is a need to perform manual discovery in centralized API repositories or using general purpose search engines, both come with a hefty amount of drawbacks. Semantically described RESTful services are not very common and use heterogeneous description formats and vocabularies for annotations, which does not make the task of automatic discovery easy for clients and agents, which have to be hardcoded with some domain-specific ontologies and description format-dependant code in order to allow the interoperation of the services in question. Besides, fully semantically described Web services have never really left the academic research area to gain wide adoption on the Web for the reasons explained above.

In short, the discovery of services and functionality on the Web scale has either been achieved by (1) description format-dependant algorithms which discover only the services that use that description format and consequently suffer from its disadvantages or (2) general-purpose discovery algorithms that do not fully exploit the potential and power of the technologies of semantic Web and Linked Data (semantic interlinking and annotations on links, data and operations).

At the same time, service selection plays an important role in helping the users to automatically choose, from numerous and varying candidates, the services that will carry out the tasks required by their request. Selection is based on the non-functional descriptions and the Quality of service attributes as well as the user's context, preferences and settings. In a distributed environment, where the entire set of services to choose from is not known at the beginning of the discovery process, it is necessary to continuously compare the new services discovered to the ones the client has already selected in order to determine whether or not the freshly accessed services are better suited to answer one or more of the user's task.

In chapter 2, we presented a description model that allows the semantic annotation of service aspects such as functionality provided (operations) and interlinking. We discussed its simplicity, modularity, extensibility and the reasons we think it would be a beneficial approach to establish as a standard for describing RESTful APIs on the Web. We believe that if a similar model were to be largely adopted by the RESTful linked service providers, it would allow easy, scalable and robust discovery of functionality on the Web. The adherence of service providers to this description model and to quality of service models would also provide a solid base upon which a fair market of services would be offered to the users on the Web to allow automatic discovery and composition of functionality on a semantic Web of services.

In this chapter, we introduce a set of algorithms for discovering and selecting se-

mantically described resources. As we work in the context of resource oriented Web, the discovery of new resources that can potentially participate in answering the user's request is done progressively following the principle of HATEOAS. We rely on semantic annotations developed in previous work [Bennara et al., 2015b] to describe resources. Such descriptions include data-related semantic annotations, available HTTP operations, relations with other resources and non-functional service properties. We also rely on breadth-first search algorithm combined with a skyline-based approach [Borzsony et al., 2001] to select the appropriate resources while maintaining acceptable performance.

The remainder of this chapter is organized as follows : section 3.2 presents the state of the are of service discovery, section 3.2.2 presents the state of the are of service selection, section 3.3 details the important role the descriptions play in guiding the processes of discovery of functionality and selection of services that provide the functionality, section 3.4 presents our contribution for the problem of discovery, section 3.5 presents our contribution for the problem of selection and in section 3.6 we discuss and evaluate the different proposals and contributions.

3.2 Related Work : Discovery and Selection

3.2.1 State of the Art of Service Discovery

Discovery of classic RPC Web services has seen a fair amount of research activity dedicated to it with a lot of solutions that allow discovery of Web services based on functionality, inputs and outputs. Some solutions propose centralized registries like UDDI, which simplified the process of discovery but proven to scale poorly especially in an open environment like the Web. Centralized solutions have only been successful in the enterprise environment, where services are exposed only to a limited amount of users within the same business.

Distributed versions of these centralized solutions have been proposed, but they did not provide the needed scalability for the increasing amount of data and users on the Web, plus they still suffered from the same problems as their centralized counterparts. This sparked the need for completely different class of solutions where the services are completely distributed, no central repository or registry exists. These solutions rely on the interlinked nature of the Web and huge amount of data sources and services that refer to each other using hypermedia links.

The emergence and spread of RESTful APIs and the advent of the Linked Data as a standard for publishing structured data on the Web has prompted the need to create

service descriptions that comply with the constraints imposed by REST architecture and Linked Data. Researcher have started realizing the potential of integrating the two standards together in order to achieve the automated discovery and composition of functionality on the World Wide Web. However the heterogeneous description formats proposed by the research community, and the non adherence of the service providers to any of them at a significant scale, has hampered this ambition and researchers have started proposing discovery solutions based on their own description formats.

In the following, we present some of the solutions researchers have proposed in order to address the problem of service discovery on the Web, and in RESTful resource oriented Web in particular.

3.2.1.1 Centralized Discovery of Classic Web Services : UDDI

Universal Description Discovery and Integration [Clement et al., 2004] or UDDI was the standard for the discovery of Web services based on the XML stack of technologies (SOAP and WSDL). It was established as a standard by OASIS [67]. It was used as a centralized registry for businesses and enterprises to publish the WSDL descriptions of the interfaces of the Web services they provide.

The discovery of services using UDDI was done in a centralized manner, meaning that all discoverable services were available at once in the registry. The client performing the discovery could do so based on the interfaces described by the WSDL descriptions together with the other information regarding the service provider and its context. The discovery process was relatively simple, given the centralized nature of the registry, but the diversity of the results was limited by the fact that new service providers have to register themselves before making their services available for discovery. This allowed popular and influential services to gain monopoly on certain service applications and slowed the emergence of new service providers for those applications.

UDDI suffered from a considerable amount of drawbacks, hindering its applicability on the Web, most notable of which are its non scalability in a huge information environment and also its technical problems such as being a Single Point of Failure, meaning that if the access to the centralized registry is blocked the discovery of the services within it becomes impossible. Some semi-decentralized solutions where clusters of UDDI registries are exposed rather than a single massive registry, but they suffered from the same problems and introduced more complexity to the solution such as synchronisation between the different separate registries and registry consistency.

67. `https://www.oasis-open.org/`

3.2.1.2 Discovery of RESTful Web Services

RESTdoc

RESTdoc [John and Rajasree, 2013] is a description solution that combines multiple micro-formats in order to semantically describe RESTful resources. For a more detailed discussion about the description model of RESTdoc, refer to the state of the art section in chapter 2. The discovery mechanism proposed by the authors of RESTdoc allows locating related or similar services given a description of a particular service. It distinguishes two different aspects of RESTful service discovery problem :

— The discovery as a client concerns the client-side browsers. This is the mode of discovery we address in our research, it describes how a machine client can discover new services and resources using the representation and the description of the service being currently browsed. It relies on HTML Link element on a Web site in order to point to other resource descriptions. Since RESTdoc descriptions use microformats and are embedded directly to the service's representation, the client is guided directly by these annotations in order to determine whether or not the links being provided are relevant to the current search.

— The discovery as a service which is is the ability for a service to access and link to other related resources in the same application domain. This mode of discovery can be used by services themselves in order to locate other related or complementary services that can be linked to in their representations. This can also be used by services that require other services to perform their functionality, but the availability of the services they usually cooperate with is not guaranteed. This is achieved by building a sub graph of related resources and storing it for later access where the graph is traversed by the discovery mechanism and the services that are needed are called upon.

The discovery solution provided by RESTdoc relies on the semantic annotations attached to the HTML elements in the services being described. This means that if a service on the Web does is not described according the description format advocated by RESTdoc this discovery mechanism cannot be applied. In other words, although the discovery solution describes a fully peer to peer mechanism, it is dependant on the description format and is thus not applicable to the entirety of services on the Web. In addition to that, RESTdoc description proposal imposes service providers to provide the semantic annotations on the actual service and resource representations, which is not an easy and incremental approach to describe the services, and thus their visibility to general purpose discovery algorithms will be limited.

RESTdesc

RESTdesc [Verborgh et al., 2011a] is a work about semantic description of Web APIs based on the Notation3 RDF syntax. The purpose of the description is to allow for an efficient way to discover the features that Web APIs offer, based on their functionality instead of the classic approach of input-operation-output. It uses operational semantics of Notation3 in order to allow a flexible discovery thanks to the powerful and robust N3 reasoners.

Similarly to RESTdoc, the descriptions of services and resources in RESTdesc allow the discovery algorithm to access the semantic annotations on different service elements in order to determine whether or not the service matches the search criteria. The important difference here, is that RESTdesc descriptions are external to the actual service representations. This means that general purpose discovery algorithms can be easily repurposed in order to use RESTdesc descriptions as the determining factor in the discovery process. This is made possible by the fact that such descriptions can be integrated with services easily, and are made available through different mechanisms such as the HTTP method OPTIONS, for example.

Furthermore, RESTdesc is all about explicitly describing functionality. This simplifies the discovery algorithm by comparing the functionality description published in the service's description with the functionality required in order to to perform the task in question. Also, RESTdesc describes preconditions and postconditions of the said functionality, meaning that if the preconditions of the application state meet with the preconditions of the service being discovered, it can be called upon to make the transition to the next application state where the postconditions of the functionality become new preconditions for the next iteration, and so on.

Although adding RESTdesc's functionality oriented descriptions to services can be achieved relatively easily, the problem lies in the inherent complexity of such descriptions. Despite Notation3's logic-oriented syntax and the powerful tools built around it, the knowledge threshold and effort required to describe services using this description format is considerable and adhering to this description format on a wide scale would be costly for service providers.

3.2.1.3 Social-Based Discovery Model : LinkedWS

LinkedWS [Maamar et al., 2011b] is a Web service discovery model based on human interactions in social networks in the context of a service oriented architecture. The idea behind LinkedWS is to establish a social network of Web services where nodes are actual

Web services and edges are relations between these Web services. It describes the relationships between services on the basis of the interactions that happen between them, using a model based on social relationships between people in social media networks. In other words, the approach proposed analyses the interactions between services in the same way the interactions and relationships between people in social media networks are established, and creates relationships between services in order to facilitate their discovery once the discovery algorithm accesses a service in the network.

LinkedWS takes advantage of the advances of social media networking, that has seen a considerable and fruitful research activity in the last few years, and establishes a model to enable a scalable discovery of service functionality. It promotes the use of relationships between services such as substitutions (services that offer the same functionality, that can be substituted to each other) and collaborations (services that can collaborate together to provide a value-added composite service). This supports both the discovery process and the selection process. In case the automated client want more candidates for a given task, it can explore the similar services in order to discover more services that can perform the task in question, and if the selection process filters out a certain service because its non-functional properties do not comply with the user's context, services that can be substitutes for it may have the desirable non-functional properties to satisfy the users needs. Also, in order to discover more services that perform other tasks, which are potentially complementary and are usually performed in collaboration to answer certain popular user requests, the links to complementary services are explored. Complementary services are services that are usually called upon in the context of frequent requests, such as booking a flight to a given city and a hotel room in the same city. The establishment of such links with semantic annotations indicating their relationships to other services, which can potentially be used in collaboration with them, is a major enabler for a scalable and distributed discovery of services in a resource-oriented Web.

What is really important about this proposal in the current chapter is the categorization established on exposed functionality. We use a similar model in order to annotate semantic links in our service descriptions, and we build our discovery solution upon these semantic annotations in order to propose a scalable and semantically enabled discovery algorithm.

3.2.1.4 Graph Discovery Algorithms

Exploring very large graphs such as the Web requires efficient algorithms in order to have acceptable response times. As we are discovering resources on the Web, the efficiency of the exploration algorithm is one of the most important elements of our research

work. Practically the classic algorithms are not used as such because they may result in important response times due to the Web size. Instead, variants of these algorithms are used with specific parameters (often limiters) in order to yield reasonable response times and acceptable results. The most known examples of algorithms are Breadth First Search and Depth First Search algorithms. Other algorithms include variants of these two with limiting parameters, for example limiting the depth of the search (number of consecutive edges counting from the root) also known as depth-limited search or limiting the total number of nodes accessed during the whole process [Russell et al., 1995] [Förster and Wattenhofer, 2012].

According to [Najork and Wiener, 2001] the Breadth-First Search graph traversal algorithm yields high-quality pages early on in a crawl. In other words, the most relevant pages/resources to the search are discovered early on in the process. In our work, this means Breadth-First Search finds the most relevant resources to answer a user's request by finding multiple (or single) resources that can perform the tasks needed in order to answer the request. In addition to that, Breadth-First Search is a very natural search strategy in the context of Web. Also, compared to other efficient search algorithms, it has a relatively low computational cost for a large scale graph such as the Web.

3.2.1.5 Synthesis

As centralized solutions for discovery of services on the Web has proven to be insufficient to cope with the huge and increasing amount of service being exposed, new approaches began to emerge to enable scalable discovery of functionality on the Web. Furthermore, the emergence of Linked Data and semantic Web compliant service descriptions has opened new challenges to allow a semantically-enabled service discovery. The bottom line from our bibliographical RESTful linked service discovery is that the formalism of service description highly influences the efficiency of the discovery process. Although some proposals specify description models that can more or less be easily integrated with existing services, semantically-enabled discovery remains a difficult scientific lock to address due to the heterogeneity of the description formats used by the services on one hand, and the slow adoption of said description formats on a large scale.

The application of efficient general-purpose discovery solutions to allow a semantically enabled discovery depends on the simplicity of the description format used, this is why lightweight, modular and extensible description formats are considered superior to complex and verbose ones. The description model we proposed in chapter 2 is specifically aimed at simplifying the application of these general-purpose graph traversal and discovery solutions on services described using our proposal.

The contribution we propose in this chapter relies on exploring the semantic annotations over the links between resources, which are found in descriptors, in order to guide the discovery process into the links with the most potential to match with the request concepts. We reuse the Hydra core vocabulary in order to establish these descriptions. However Hydra does not provide a good support for semantic annotations over links and operations. We extend hydra in order to allow resources description to have semantically annotated elements that can be exploited by discovery, selection and composition algorithms in order to enable a completely automated process to answer user's requests. The simplicity of our solution lies in the separation between resource representation and description as well as the separation between links and operations in the descriptions. In order to discover resources that can answer the user's request, the generic client has to start exploring the description of the resource given as an entry point by the user. Based on the semantic annotations given by the description, the decision making of (1) whether or not to account the current service in the final composition and (2) what are the next resources to explore is easy to establish.

3.2.2 State of the Art of Service Selection

Selection of services has been an active research topic since the spread of Web services in the World Wide Web. Quality of service data about different services and service providers has been the main deciding factor in selection algorithms proposed, together with the user preferences and requirements. Capturing these two aspects has also been a topic that interested researchers that worked on service description and Web client interactions. A lot of models for quality of service have been proposed as well as models to capture user preferences when it comes to service usage. In the following we are going to present key research proposals that address the problem of quality of service modelling and service selection.

3.2.2.1 Quality of Service in Service Oriented Web

During the last years, the description of the non-functional properties of services as well as quality of service attributes has seen a considerable research activity by the research community.

Ran [Ran, 2003] proposed a model for QoS in Service oriented Web that divides QoS attributes into several different categories. This is one of the efforts to provide a complete set of attributes that describe the QoS aspects of a service. It is designed for the service-oriented architectures and particularly the Web. It proposes an extension to UDDI in order

to support non-functional properties of Web services in the descriptions published on the registry. Also, the authors propose the role of *certifiers*, which take the responsibility to verify the QoS claims of different service providers, similarly to how rating agencies rate the quality of real life services in various domains.

Although the quality of service model proposed here is aimed for the classic Web services, it is very much relevant to RESTful services in the context of semantic Web. The non-functional aspects of services need to be described semantically however, in order to allow the automated clients and agents to reason about their meaning and allow the comparison with the user's requirements. A semantically-enabled QoS model based on the model proposed here could be established in services descriptions in order to automate the selection process.

AgFlow [Zeng et al., 2004] is a solution that enables quality-driven discovery and composition of Web services. The authors propose a model to collaboratively evaluate the overall quality of service of a composite service, where each task is potentially performed by a service from a different provider. It proposes two interesting different approaches to select Web services for a given task within a composite service. The local optimization approach, on one hand, suggests to perform the selection of the best candidate for a given task solely in the context of that task. In other words, the other tasks and the services to be selected for them are not taken into account when filtering out the candidates that can perform a given task. In this approach the selection is left to the last possible moment until all the available candidates are known, and then the selection process is executed. Although the QoS is maximized for each individual task, the overall QoS, which is calculated using the proposed model, can be suboptimal. In the global planning approach, on the other hand, the selection is done for each task individually but by taking into account the other tasks. This ensures that the overall QoS of the composite service is optimal This also allows the automated client to readjust the selected candidate for a given task while discovering new services.

Using the global planning approach to carry out the selection process is an interesting take on the selection problem, but the computations to ensure such criteria can be very costly when the composite service involves a large number of tasks.

3.2.2.2 QoS-Based Web Service Selection

The problem of selection of Web services is a part the composition process that involves the QoS aspect to choose the most suitable services for the user. Finding the optimal solution for this problem with multiple QoS constraints is a NP-hard combinatorial optimization problem [Yu et al., 2007]. This problem can be modelled as follows : the user

emits a request that requires several Web services to be answered. The solution for this request is divided into several tasks, every task can be performed by a single Web service. For every task, the client can potentially discover several candidate Web services. Finding a solution amounts to choosing the best candidate for each task in order to obtain the highest overall QoS, or choosing the best candidates to ensure the maximum overall QoS for the composite application. In our research, we follow a different setup where the search space of candidates progressively discovered by following links between resources.

Wang et al. [Wang et al., 2011] propose an approach for selecting services based on Generic and Domain-related QoS attributes (DGQOS). Generic QoS attributes (GQoS) can apply on any type of Web service. Domain QoS attributes (DQoS) apply only on a certain class of Web services. The authors define evaluation models for DQoS and GQoS attributes, which help calculate the overall QoS of composite Web services based on its components. They use the C-MMAS (Cultural Min-Max Ant System) algorithm in order to solve the selection problem.

Alrifai et al. [Alrifai et al., 2010] propose a solution for selection Web services based on the skyline approach. The goal is to identify for each task the services that will never be part of the final solution simply because they are outclassed by another candidate service in every QoS-related aspect. The authors try to keep the set of candidates as small as possible in order to apply constraint optimization algorithms to obtain the best solution. They also propose a solution to further reduce the size of the set of candidates by identifying representative candidates that replace a subset of candidates that have similar QoS parameters.

3.2.2.3 Synthesis

Many models to capture the non-functional properties of services have been proposed in the literature. Semantically-enabled selection can be achieved on the basis of such models, by allowing the semantic annotation of the QoS attributes exposed in service descriptions. Also, many solutions have been proposed to filter the discovered services and choose individually or collectively the set of candidates for each task in order to achieve the maximum satisfaction of the user's requirement.

In the context of RESTful Web services, the set of candidate resources to be selected is progressively discovered, since the graph of the possible service candidates is traversed by the discovery algorithm, and is not visible from the start. In fact, the setup of a centralized registry for all resources is not adapted to the distributed and large scale nature of the Web. Therefore, the discovery process gradually discovers resources that can fulfil a given task and does not have all the candidates until the end of the algorithm. Plus, the space

of solutions expands as the discovery algorithm finds new candidates which increases the computational cost of the selection process. Here, we face two possibilities. The first one is to run the selection algorithm while the discovery algorithm is exploring the resources on the Web. When a new candidate is discovered, it may be a part of the final solution. The second possibility is to run the selection algorithm at the end after having all candidates for every resource, the selection algorithm is run a single time after the end of the discovery algorithm.

Also, the criteria for the selection can be defined on the local (per task) level or the global (the entire process) level. A middle-ground would be dividing the process into sub-processes where the dependencies between the tasks are stronger inside a sub-process than between tasks in different sub-processes. Regardless, there is a trade-off to be made between computational costs and best overall QoS for the composite application.

3.3 The Description Role in Discovery and Selection

Our descriptor-based solution allows generic clients to crawl from one resource to another in order to select interesting resources to answer the user's query. However, due to the huge number of resources on the Web, there is a need to improve the discovery algorithm that we use (i.e. BFS [Kozen, 1992]) to only select the most interesting resources. The vocabulary we use to implement the descriptor concept is Hydra core vocabulary [Lanthaler and Guetl, 2013]. We introduce semantic annotations on descriptor links, and extend the BFS algorithm to take advantage of these annotations. The semantic annotations will guide the algorithm by excluding irrelevant links to the current application. Fig 3.1 illustrates the semantic annotation of descriptor links. Our semantic annotation is inspired from existing work [Maamar et al., 2011b] to define the properties that link resources to each other. We define that :

— Two resources are `similar` if they provide functionally substitutable services, sometimes varying in terms of non-functional properties.
— Two resources are `complementary` if they can be combined in the same process to answer user's needs, for example : a flight booking service and a hotel booking service in the context of a trip.
— Two resources are `incompatible` if they cannot be involved together in the same process because of a given reason, for example : the eBay online seller could decide not to work with the UPS delivery company.

FIGURE 3.1 – Descriptor example with annotated links

3.3.1 Descriptive information guiding the discovery and selection

Our solution builds a generic client that interacts with the resources through their respective APIs. The client software program needs to be able to automate the process of composing the functionality of the three resources of the scenario to answer the user's query. This includes the discovery of the resources. The maximum number of `similar` links to be operated can be limited in order to increase the performance of the BFS algorithm. However, this will limit the choices given to the user. A compromise between performance and result diversity is to be established using this parameter.

We also propose a solution inspired by the weight-based approach presented in [Ge and Qiu, 2008] in order to :

— Sort the links on a resource description in order to guide the discovery algorithm while exploring similar links.

— Sort the results obtained after positive matching with a query concept

In other words, the set of similar resources inside a resource descriptor are sorted from the most similar link into the least similar one. Based on the query nature, the discovery algorithm starts exploring the most similar resources if the priority is to find more alternatives to the current resource or the least similar resources if the priority is to find more complementary and diverse resources. Many formulas to calculate semantic distance have been proposed in the state of the art [Ge and Qiu, 2008, Hau et al., 2005]. The one we adopt in our work is weight-based formula proposed in [Ge and Qiu, 2008] because it can be directly used with our approach without any further calculation of additional parameters. Note that the work of annotating links and sorting them is not done during the discovery.

We consider a user request that involves a set of tasks, each task is represented by an ontology concept. The set of tasks is organized as a workflow that represents the series of actions that the client needs to perform in order to deliver the result to the user. The Web resource that can fulfill a specific task is not unique due to the nature of the Web and the client can discover many candidate resources to fulfill the given task. However, not all these candidates match the user requirements in terms of QoS, and therefore a

selection phase is needed in order to determine the most suitable candidate for each task.

$T = \{OntologyConcept\}^N$ is the set of tasks in the workflow, where N is the total number of tasks.

$C_i = \{(URI, operation)\}^{m_i}$ is the set of candidates for a task t_i, where m_i is the total number of candidates for the task t_i. The candidate number j for the task t_i is therfore c_{ij}.

Finding a solution for the user's request amounts to finding the set of candidates c_k for each task t_k, where every candidate meets the hard constraints for the user at least (and preferably the soft constraints) and the overall QoS of the set is the best among all combinations. We define hard constraints as conditions that must be fulfilled by a discovered resource otherwise it is not eligible for the task. On the other hand soft constraints are optional conditions that are not necessary to select a candidate for a task.

As we have opted for a hypermedia-driven approach for exploring the Web, we need an on-the-fly selection strategy in order to be able to select relevant resources along the discovery process. In the remainder of this section, we detail a "select while you discover" strategy to enable on-the-fly selection.

3.3.2 A minimal QoS model for Web resources

To present our solution, we rely on a minimal model based on [Ran, 2003] in order to describe some important non-functional properties of a Web resource. QoS :

{

Performance : [0-10],

Availability : [0-100],

Reputation : [0-5]

}

3.3.3 QoS-based resource selection problem specification

The problem of selection of resources is part of the composition problem. This problem has been proven to be NP-Complete. The process of reasoning on the user's request, as described in the scenario, can be assimilated to a Web service selection problem. Every task of the composite solution can be fulfilled by a resource, that has to be discovered. Multiple resources can be candidates for a single task. Tasks are semantically identified by the concepts the reasoner infers after analyzing the user's request. We start with N

tasks and for each task i we have M_i candidates. The problem is to identify the set of candidates S where each candidate s_j fulfills the task t_j and the overall QoS of the set is the best according to the user's preferences.

3.4 Contribution : HATEOAS-Based Discovery Algorithm

The discovery algorithm [Bennara et al., 2016b] details are given in Algorithm 1.

The algorithm takes as input three parameters :

— `conceptList` (array) : contains the list of concepts that describe the operations needed in order to answer the user's query.
— `currentLink` (string) : contains the URI of the resource being processed.
— `similarLimit`(int) : is the maximum number of similar links per resource to be taken into account by the algorithm.

The algorithm returns as output the `result` array which contains all the pairs [`concept`,`URI`] where the resource identified by `URI` can perform an operation that semantically matches the paired `concept` classified by semantic distance from the query concept.

The set of variables used in this algorithm are the following :

— The `bfsQueue` is the queue that contains the ordered set of URIs for the next nodes to be explored by the BFS algorithm.
— The `visited` array contains URIs of resources already traveled. This variable's main objective is to prevent loops if the graph is cyclic. Further improvements on this part of the algorithm are possible in order to obtain better performance.
— The `similarCount` variable introduced in line 12 counts the number of `similar` links that are inserted in the BFS queue to be traveled. This counter cannot exceed `similarLimit`.

The algorithm consists of a main `While` loop. The exit condition is verified when there are no concepts to look for or no further resources in the graph to travel or when a certain amount of time passed since the beginning of the loop (timeout). Each iteration of this loop discovers a single resource whose link is `currentLink`. The algorithm verifies if it has not been visited yet, if not it is marked as visited. If the resource has not been processed yet, the algorithm gets its descriptor then checks if any of the operations provided by the resource is annotated by one of the remaining concepts. If so, the concept along with the resource URI are inserted into `result` then the concept is removed from `conceptList`. After that, the algorithm inserts the URIs of the related resources into `bfsQueue`, while

Algorithm 1: BFS-based discovery algorithm

```
   Input: conceptList : array of string
   Input: currentLink : string
   Input: similarLimit : integer
   Output: result : array of string
 1 bfsQueue : array of string
 2 visited : array of string
 3 while not conceptList.empty() and not bfsQueue.empty() do
 4     if not currentLink in visited then
 5         visited.insert(currentLink)
 6         Descriptor descriptor = getDescriptor(currentLink)
 7         foreach operation in descriptor.operations do
 8             foreach concept in conceptList do
 9                 if conceptMatch(operation.annotation, concept) then
10                     result.insert([concept, currentLink])
11                     conceptList.remove(concept)
12         similarCount : integer = 0
13         foreach link in descriptor.links do
14             if link.annotation = IsComplementary then
15                 bfsQueue.insert(link)
16             else
17                 if link.annotation = IsSimilar and similarCount < similarLimit
                    then
18                     bfsQueue.insert(link)
19                     similarCount = similarCount + 1
20             //and if it is incompatible we do not take it into account in the first
                place
21     currentLink = bfsQueue.next()
```

respecting the fact that similar resources links inserted cannot exceed `similarLimit`.

Setup	Advantage	Drawback
Selection of the first solution that matches the user's QoS profile	Fast solution	Low overall QoS, may not match with the user's soft constraints
Selection of the best candidate for each task	Selection of the best candidate for each task	May not be the best solution to obtain the best overall QoS
Selection of the best solution, by exploring all combinations	Ensures the best solution amongst all combinations	Slow solution, exponential processing time
Selection of the best solution while eliminating irrelevant candidates using skyline approach	Ensures the best solution amongst all combinations, while having less candidates to work with	Still a relatively solution but a lot better than naive exploring of all combinations

TABLE 3.1 – Different on-the-fly selection setups

3.5 Contribution : On-the-Fly Selection Algorithm

The selection process is executed at the same time as the discovery goes on. Each time a resource is discovered, the selection algorithm is run in order to verify if this new resource can be the best candidate for its task among the other previously discovered resources for that same task while, at the same time, making sure the set of selected candidates for all the tasks verify certain conditions (best overall QoS matching with user's profile, compatibility between resources, etc.).

In table 3.1, we present four different setups to enable the on-the-fly selection (select as you discover)

The general selection algorithm takes as input the set of all candidates T for each resource, the current set of best candidates s and the new candidate c resource as well as the user's QoS preferences *qos* and returns a new set of best candidates and updates If it is selected the set of best candidates is updated with the better new solution. For this purpose, we consider a discovered candidate as an object composed of two attributes resource URI and HTTP operation (`uri` and `operation`) plus the concept that matches is with the operation (`concept`). In the context of the algorithms presented below, we define the concept of domination as follows : a candidate c_1 dominates another candidate

c_2 if all of c_1's QoS attributes are equal of better compared to c_2's QoS attributes.

Algorithm 2 [Bennara et al., 2016a] shows the optimized global planning method. This algorithm eliminates the candidates that will not be part of the final solution before reevaluating the solution. If the new candidate cannot be added to the set of candidates (is irrelevant), the algorithm does nothing and skips this iteration.

Algorithm 2: On-The-Fly optimized selection algorithm

```
   Input: s : array of Candidate
   Input: c : Candidate
   Input: qos : QoSprofile
   Input: T : array of array of Candidate
   Output: s : array of Candidate
 1 var s2 : array of Candidate = s
 2 // If the new candidate matches user requirements :
 3 if QoSmatch(c, qos) then
 4     // add c while removing irrelevant candidates
 5     // if c is irrelevant quit if (skyline(T, c) = true) return
 6     // and verify if there is a new best solution :
 7     for i = 0 to T.size do
 8         for j = 0 to T[i].size do
 9             s2[i] = T[i,j]
10             if QoScalculate(s2, qos) > QoScalculate(s, qos) then
11                 s = s2
```

Algorithm 3 shows how to insert the new candidate and how the irrelevant ones are removed right after.

N-Periodic Selection

Launching the selection every time we have N new candidates for a given task can reduce processing time for the selection phase, with the skyline based setup. Note that the number of new candidates for a given task t is n_t where $\sum_{i=1}^{M} n_t = N$ (M being is the total number of tasks).

Algorithm 4 shows how to apply the skyline approach to reduce the size of candidates for a given task t when n_t new candidates are discovered in each iteration of the selection process, instead of only one.

Algorithm 3: Inserting the new candidates and removing the irrelevant ones using the skyline approach

```
   Input: c : Candidate
   Input: T : array of array of Candidate
   Output: T : array of array of Candidate
 1 var x : type = init
 2 // if c is not dominated by any other candidate for the same task
 3 for i = 0 to T[c.concept].size do
 4 |   if dominate(T[c.concept][i], c) then
 5 |   |   return true
 6 // insert it T[c.concept].insert(c);
 7 // remove candidates dominated by c foreach c2 in T[c.concept] do
 8 |   if dominate(c, c2) then
 9 |   |   T[c.concept].remove(c2);
10 return false
```

We know that the set of old candidates for the task t is a skyline i.e. no old candidate in $T[t]$ is dominated by another one in the same set. The first step is to eliminate the new candidates in $(N[t])$ that are dominated at least one candidate of the same set. After that, we eliminate the new candidates (which is now a skyline) that are dominated by at least one old candidate $(T[t])$. Next, we eliminate old candidates dominated by at least one new candidate. Now we know that no candidate in $T[t]$ or $N[t]$ is dominated by any other candidate in the two sets. Finally, we merge the two sets in order to obtain the skyline of candidates for the task t.

3.6 Discussion and Evaluation

The resources composing the services previously presented in the scenario are implemented using Java TM Servlets using Jersey framework 7. We use Apache Tomcat 8 as a server-side software in order to accommodate our resources. The demonstration Web page can be found here : `https://liris.cnrs.fr/~mbennara/doku.php?id=medi2016`.

We show the number of traveled nodes as well as response time (in milliseconds) gain compared to the raw BFS algorithm respectively in Fig 3.2 and Fig 3.3. Each column represents a separate query that involves an increasing number of resources in the Web. We get better response times for the same request with the enhanced algorithm because it

Algorithm 4: Selection of n resources at a time instead of one

```
Input: N : array of array of Candidate
Input: T : array of array of Candidate
Output: T : array of array of Candidate
// apply the skyline on the set of new candidates first
for i = 0 to N[t].size do
    for j = i+1 to N[t].size do
        if dominate(N[t][i], N[t][j]) then
            N[t].remove(j);
        if dominate(N[t][j], N[t][i]) then
            N[t].remove(i); break;
// remove new candidates dominated by old ones
for i = 0 to N[t].size do
    for j = 0 to T[t].size do
        if dominate(T[t][j], N[t][i]) then
            N[t].remove(i); break;
// remove old candidates dominated by new ones
for i = 0 to T[t].size do
    for j = 0 to N[t].size do
        if dominate(N[t][j], t[t][i]) then
            T[t].remove(i); break;
// merge the two new sets
T[t].merge(N[t]);
```

explores less nodes than the regular. This is due to the fact that when we travel the Web graph, we find more similar resources. The similar resources are ignored by the enhanced algorithm but taken into account by the regular one. However, this decrease in response time can also be accompanied by a decrease in result diversity.

Enabling semantic annotations on links between resources allows the automation of the discovery process. Without the semantic annotations, the discovery algorithm has to explore every link in order to search for resources to answer the user's query. Having similar and complementary annotations on links allows the algorithm to explore the requested links based on selectivity measures. The maximum number of similar links to be explored is limited. This limit determines the performances of the discovery algorithm as well as

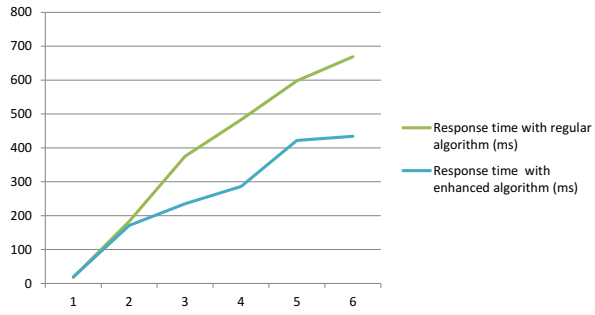

FIGURE 3.2 – Response time in ms

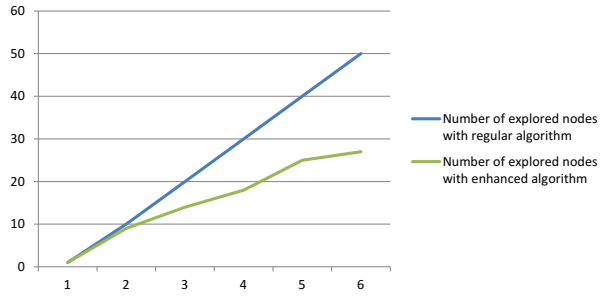

FIGURE 3.3 – Number of explored nodes

the diversity of the results obtained. The lack of diversity is due to the possibility for similar resources to contain links into useful complementary resources. Sorting the similar resource links in the description is important in order to optimize the discovery algorithm. Depending on the user's query, the discovery process will prioritize the most or the least similar links while taking into account the similar limit as well.

Executing the selection algorithm every time a new candidate is discovered can hinder the processing performance to answer the user's request. With the skyline-based solution the overall execution time can be optimized through waiting for N new candidates to start the selection.

Lets suppose the number of new candidates for each task t is m_t, where $\sum_{i=1}^{N} m_i = n$ where N is the number of tasks. Let us suppose the number of the old candidates for each task is l_t.

In the worst case (i.e no new nor old candidate is dominated by another), the number of iterations is exactly the same with One-periodic or N-periodic selection : $2l_t m_t + \frac{m(m-1)}{2}$. But in the general case, the number of iterations in N-periodic selection is lower. Indeed, we eliminate the irrelevant candidates in the set $s1$ of the newly discovered n candidates to obtain a set $s2$ of $m \leq n$ candidates. After that, we consider $s2$ and eliminate the candidates that are dominated by at least one element of $T1$ to obtain $s3$ with $|s3| \leq |s2|$. Next, we consider the complete set of candidates $T1$ and eliminate the candidates that are dominated by at least one element of $s2$ to obtain a new set $T2$ where $|T2| \leq |T1|$. Finally we merge $T2$ and $s3$ to obtain the final set $T3$ that represents the whole set of candidates without irrelevant candidates.

Selection algorithms of the skyline set of services

After reducing the number of candidates with the skyline algorithm, we need to choose the best solution for selection. In our contribution we show the naive combinatorial algorithm in order to explore the reduced space of solutions. We use a double loop in order to explore the two dimension array of candidates. There are some optimized algorithms [Wolsey and Nemhauser, 2014] specifically aimed at obtaining better performances for this class of optimization problems.

In some cases, the size of the set of candidates is very large that, even with the algorithms we proposed, the solution can not be obtained in a reasonable amount of time. Some solutions have been proposed to resolve this problem, such as the representative skyline services proposed in [Alrifai et al., 2010].

3.7 Conclusion

In this chapter we propose an annotation of Web resource descriptions based on a social model that relies on similar and complementary relations. These annotations provide information for the discovery process in order to respond to the user's request faster and more accurately. Then, we provide a semantically-enhanced BFS-based algorithm to discover resources. It relies on the semantic annotations in order to determine whether a resource is worth exploring.

Future work includes exploring advanced heuristics to reach a better compromise between performance and result diversity. We envision to extend our model to support quality of service aspects in order to further enhance the discovery and selection processes. We aim also to enable an automatic service composition process in order to fully automate answering users' requests.

In this chapter, we propose a skyline-based approach to enable Web resource selection. We show that a solution based on the HATEOAS principle, where we select the Web resource candidates along the discovery stage, is more efficient for selection than a classical solution that consists in waiting for discovery results before the selection stage. We rely on a minimal QoS model to demonstrate our approach. We provide four different setups in order to satisfy the user requirements according to the QoS profile and preferences. We enhance the performance of our solution with a skyline-based algorithm in order to reduce the set of candidates for a given task and demonstrate that it gives the same output as with a fully combinatorial algorithm but with less candidates and therefore less overall computational time.

As future work, we envision to consider constraints between candidates for different tasks while running the selection process. In other words, the set of candidates for a given task can be different depending on the chosen candidate for other tasks and also on the user's preferences.

Chapter 4

RESTful Linked Service Composition

Contents

The use of RESTful Web services promotes stateless service interaction and decentralized hypermedia-driven discovery and composition. However, there is a need for models and tools to drive user interaction as well as description, discovery and composition of RESTful services. In this chapter, we provide a solution to help users manage, share and discover workflows of RESTful Web services. We annotate RESTful Web services with semantic information, and introduce the notion of *composition directory* as a Web resource that assists a user in sharing, managing and discovering workflows. Users' composition directories form a decentralized repository of service workflows connected by hypermedia links.

4.1 Introduction

The Web has moved from a Web of documents to a distributed application platform where applications are exposed as Web resources, as witnesses the growing number of available APIs[68]. Leading research topics are related to discovery, composition and invocation of Web resources via their API. In addition, the emergence of semantic Web technologies gives the opportunity to improve the use of APIs with semantic annotation over Web resources. Semantic annotation helps to drive the interaction with APIs by providing explicit description of domain-specific information about resources.

Another key concept that drives today's Web is distributed affordance. Affordance is the ability for a user to use a Web resource. The idea is to dynamically create affordance based on the information already present in a resource representation, with knowledge from distributed sources [Verborgh et al., 2013]. Distributed affordance combines the information on resources and the knowledge on service providers, as well as user profiles in order to generate possibilities for manipulating Web resources. It should allow client-side software to dynamically drive the interaction with Web resources, therefore service providers do not have to anticipate user interaction and avoid deploying static business processes that constrain users. Using this information, affordances can be created dynamically. Users can chose to execute one affordance, this will result in the application of the service offered by the provider on the representation of the resource.

In order to enable distributed affordance, Web resources must be semantically described, and user agents needs to be able to exploit such descriptions. In this chapter, we build on previous work to semantically annotate Web resources [Bennara et al., 2014b] and facilitate resource discovery and browsing. We introduce the concept of composition directory to help users manage and share compositions, and show that the breadth-first search algorithm can be used in this context to crawl and discover resources according to a composition workflow the user provides.

4.2 Related Work : Composition

Researchers are interested in the problem of describing the semantics of the sequences in the executions flows, many solutions have been proposed. The first solutions relied on syntactic descriptions of Web services in order to determine the nature of the inputs and outputs of each service call, and the orchestration of the actual composition had to be carried out manually, since there were no semantics associated with the descriptions of

68. `http://www.programmableweb.com`

services.

The rise in popularity of RESTful APIs also meant that classic service composition solutions were no longer feasible due to the different design of services in a resource-oriented Web, where the service interfaces were uniform and fixed by the architectural constraints. Also, the transition to a client-based application state rather than a service-based one meant that the composition engine in such applications is no longer the server, but the client. The adoption of the semantic Web practices in service descriptions meant that the clients could perform composition of services automatically by relying on the annotations present in the descriptions, thus eliminating (or limiting) the human intervention in the composition process.

4.2.1 BPMN

One of the most important works in this domain is the BPMN [69] [Wong and Gibbons, 2008] specification. BPMN specifies a set of flow control sequences that allows us to describe the progressing of a process. The main interest of BPMN for us is that it can be used in order to construct and store dynamic service composition processes which can be reused afterwards by another user that wants to do similar service composition. The use of BPMN relies on a Process-oriented approach rather than being Resource-oriented. This may conflict with the principles of the REST architectural style, nevertheless some concepts can be used naturally on resource oriented architectures.

4.2.2 Linked USDL

Linked USDL [70] [Pedrinaci et al., 2014b], is another work in this perspective. Unlike, BPMN, Linked USDL vocabulary has been designed especially for the service-oriented domain, making it easier to adapt for our solution. Some of the important concepts introduced by this vocabulary include : Service, ServiceOffering, InteractionPoint as well as services roles including : Producer, Provider, Intermediary, etc. which constitute the main semantic concepts of workflow control in service-oriented architectures. Linked USDL is being used in many projects, and it proved its efficient for the service community.

69. Business Process Modelling Notation
70. Unified Service Description Language

4.2.3 BPEL for REST

BPEL for REST [Pautasso, 2009a] is a work that proposes to reuse the BPEL language principles and apply it on the REST architectural style. BPEL for REST either uses the WSDL 2.0 description language without changing the current BPEL or The solution relies on the new HTTP binding element introduced in WSDL 2.0. The RESTful Web resource API is wrapped behind a WSDL document that acts as an interface between the REST Web resource and the BPEL code, using the HTTP binding with the REST resource and operation invocation on the BPEL side. The second way to use BPEL with REST resources is more direct than the first one but it requires an extension to extends BPEL in order to be able to support HTTP operations on the resource API. The main drawback of BPEL comes with its centralized approach that relies on a static composition engine, which does not fit with the HATEOAS principle. Additionally, a centralized execution process makes it less interesting in the large scale context of the Web, for which RESTful Web services have been designed. Our approach aims to use work-flows in order to enable reuse of popular compositions. This is also a flexible way of composing resources as the work-flows can be duplicated and edited.

4.2.4 Synthesis

The constraints of the REST architectural style have changed the way we discover and compose services on the Web. Many solutions have tried to adapt the existing solutions to compose classic Web services to the REST environment. A lot of the solutions rely on the description formats they propose in order to carry out an automatic composition, which hinders the applicability of a general-purpose composition solution. Also, most of the proposals provide a static composition process where the tasks can be performed by very specific services, which does not take advantage of the dynamic nature of RESTful Web services.

In the following we present the challenges we meet when establishing a solution for the composition problem. We propose a solution to represent the composition process as a dynamic workflow where each task can be performed by an abstract service. The discovery and selection algorithms we proposed in chapter 3 are used to locate and filter the concrete services whose descriptions match with the abstract service in question.

Also, since the usage of composite applications is a routinely practice we propose a solution to store, reuse and share the compositions a user has generated. In these entities, the services that perform each task can be replaced dynamically when the user's context changes or when the availability/non-functional properties of one or more services no

longer meet the user's requirements.

4.3 Contribution : Distributed Composition Directories

4.3.1 Challenges

In the context of our work, Web services are seen as Web resources that comply with the REST architectural style. The REST architectural style is based on the notion of resource as a conceptual entity that represents abstract or concrete things such as books, orders, payments, etc. Resources are identified by URIs, their state is passed to the client through representations using the adequate media type according to the principle of context negotiation. In this chapter we consider a RESTful Web service as a set of resources that provide a coherent access to the state and functionality of the software it represents [Pautasso, 2009a].

Another principle that drives the REST architectural style is the HATEOAS [71] principle. Using HATEOAS requires hyperlinks to be established between Web resources that form an open and very large graph. HATEOAS means that the discovery process is realized progressively, user agents should be able to discover other Web resources accessible from any given resource in the graph.

In this context, we identified several challenges to address, which can be summarized as follows :

— Web resource description and interlinking : resources need to be appropriate described with semantic annotations and also linked to each other with hyperlinks to enable client-side discovery (how to interact with the resource) and crawling (how to discover other resource from a given one according to the HATEOAS principle).
— Web Resource discovery : as a follow-up to the first challenge, user agents should be able to implement an efficient algorithm to crawl between resources and exploit their annotations to realize users' objectives.

In order to answer these challenges, we build on previous work to annotate resources. We introduce the notion of composition directory [Bennara et al., 2015a] to manage and share composition workflows and we show that the breadth-first search algorithm can be used to crawl through and and efficiently discover Web resources.

71. Hypermedia As The Engine Of Application State

4.3.2 Composition Directories

Based on the related work presented above, we have built our solution that promotes the concept of composition directory. Our solution must respect the following requirements in order to facilitate resource discovery and composition :

— Scalability : the increasing number of today's Web APIs makes the scalability of solutions important.
— Responsiveness : the increasing number of users generates an important load of requests on servers. We want server responses to be as fast as possible in order to handle all the requests in a reasonable time.
— Diversity : We want our resource descriptions to propose rich and diverse links to other resources in order to give them a chance for being used. In other words, the users which make a request should have different propositions rather than only popular services in a given field and thus giving the chance to less popular services to emerge if the users are interested in the services they offer.
— Dynamism : the results of the resource discovery process should not be static, in other words it should be different from one request to another, because on one hand the availability of resources involved in the request as well as the context of the request may have changed in the meantime, and on the other hand, the user context may also have changed, which implies that users might not get same results because they browsed new resources which may impact the response.
— Serendipity : the serendipity concept allows APIs to be used in a non-specific process. In other words we do not want the clients to use APIs in a deterministic way where every next API to use is already known in advance

Today, the main advances in Web resource composition are centered around the description of the resources. The focus of these advances is how to describe a resource in order to give as much information as possible to identify the nature of the resource, its activity and what type of data it exchanges. Too few efforts focus on how it links to other resources and how to follow these links, as well as how to manage and share composition workflows. The latter aspects are presented in the following in order to enable value-added resource discovery and composition.

In order to enable users to record, reuse, manage and share their composition workflows, we propose a specific resource called *composition directory*. The Composition Directory resource contains information about its owner and stores sub-resources called the composition workflows the user creates. The Composition Directory of a user links to other connected users Composition Directories. Note that this is completely compatible with the descriptor concept because the links to other Composition Directories, the Repository

and the created compositions scenarios represent the external part of our descriptor.

Figure 4.1 illustrates an example of the disposition of the different Composition Directories on the Web. Each user has its own Composition Directory, which may contain links to other Composition Directories the user chooses to link with its own.

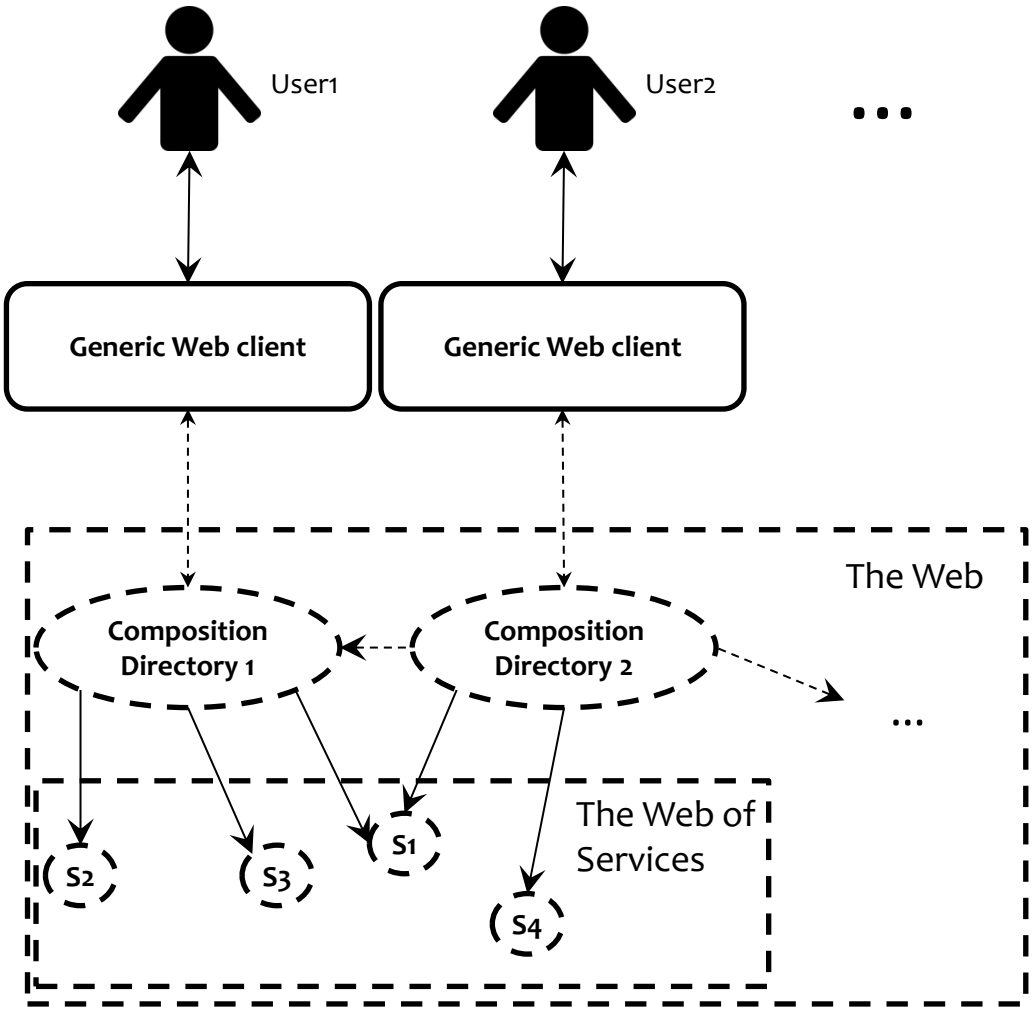

FIGURE 4.1 – Disposition of the different Composition Directories on the Web

Figure 4.2 illustrates the structure of a single Composition Directory. Each Composition Directory contains information about its owner as well as the composition workflows the user chooses to store in it, these are usually the routinely executed requests that the user wishes to automate. For example : buy a book online, book a trip with a hotel, taxi to hotel and a rented car, etc.

Figure 4.3 illustrates an example of a single composition contained within a Composition Directory. Each one of the compositions involves a number of services that are

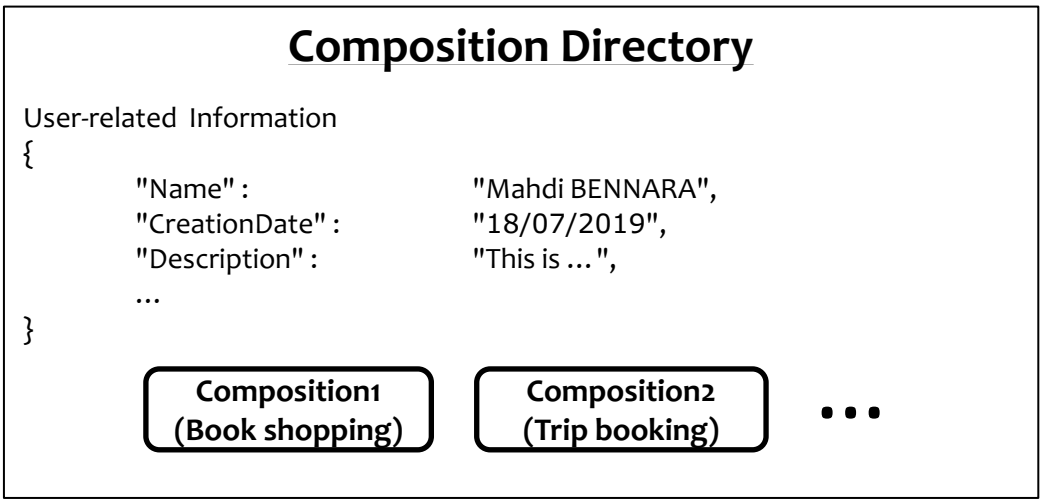

FIGURE 4.2 – Structure of a Composition Directory

abstracted on the workflow, and the candidates for each service is contained within the same composition. Also the client may retrieve other candidates from the Web or from the linked Composition Directories.

So, in order for a Composition Directory to be accessed by another one, the later should contain a simple hypermedia link to the first in its descriptor's external part. However the reusing of the compositions may be done using two possible ways :

1. The user U1 that wants to use a composition C1 stored in another user U2's Composition Directory should do a GET operation on that composition and reuse it without storing it for further uses or updates.

   Advantage : We have no composition redundancy.

   Drawback : If U2 updates C1 it may not remain interesting for U1.

2. U2 should do a GET operation on the URI of C1 stored in the Composition Directory of U2 and then stores it in his Composition directory.

   Advantage : U2 can update his own version of C1 to create newer versions that suit him.

   Drawback : We will have redundancy and problems of versioning.

The two solutions can be combined to mix their advantages and omit their drawbacks. This can be achieved by annotating the Compositions that are shared by U2, and according to its nature, type and possibility to be updated, U1 will follow the most proper solution.

We define the following API in order to qualify possible interactions with Composition Directories

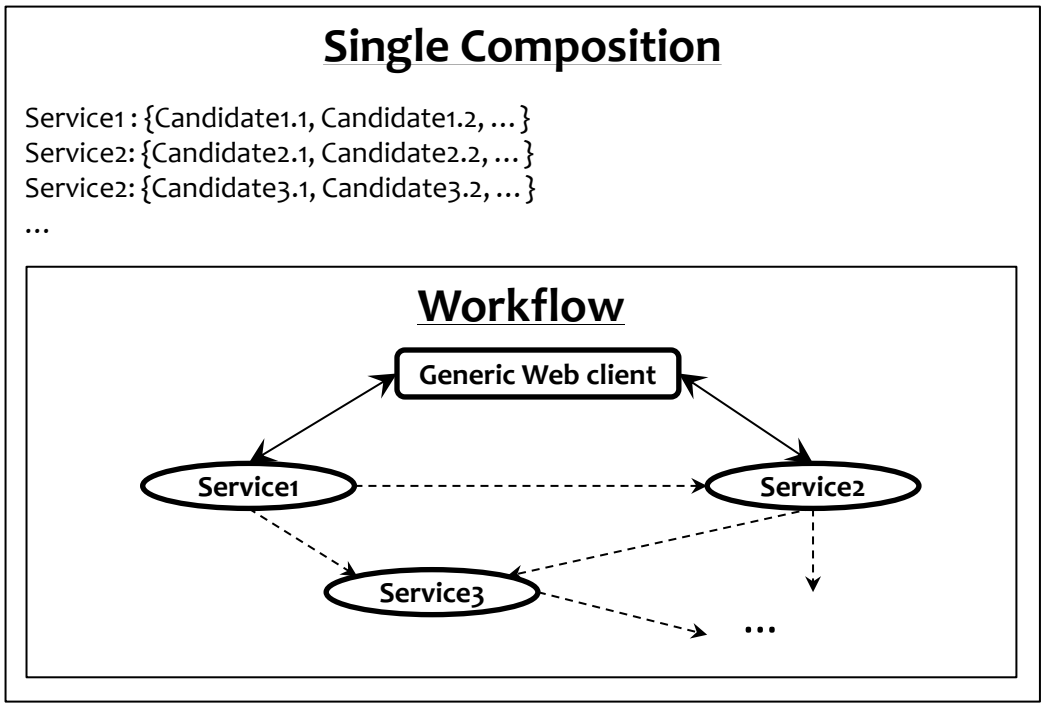

FIGURE 4.3 – Structure of a single composition contained within a Composition Directory

1. GET on the base URI of a Composition Directory should send back the information about this Composition Directory and its owner.
2. GET on the Repository of the Composition Directory should send back the set of links to every composition on the Repository.
3. GET on a specific composition URI should send back the representation of the composition. This may require an authentication and may send a 401 code (Unauthorized) in case the authentication fails.
4. POST from the user on the Repository of his Composition Directory should create a new composition. Composition attributes and its accessibility should be indicated by the use beforehand in the representation.
5. POST from the user on his own composition directory in order to add a new Composition Directory URI of another user that exposes interesting compositions for him.

This offers many advantages, first it is a scalable, decentralized and distributed solution as every user stores a part of compositions on the web, it also respects the serendipity concept as a given client may find part of the solution to the user's problem in another user's composition set. Our model includes access control features relying on HTTP authentication. We define public compositions that everyone on the Web can access from its

URI, and private compositions that are not disclosed and require authentication.

4.3.3 Discussion and Evaluation

In this section, we discuss the choices of our implementation and the impact of these choices on the challenges and the properties we want to achieve. As we are using the REST architectural style to build our solution, the respect of Web constraints is ensured by design.

Composition directories allow users to create, store and share compositions of Web resources. Compositions can be thereafter entirely of partially reused by the user himself or other authorized users. This solution allows for dynamic creation of new compositions rather than follow inflexible server-side compositions. It allows also a large scale sharing of popular compositions that users find useful and offers flexible ways to reuse and adapt compositions to user's needs. We do not rely on a central repository to store the compositions.

The scenario illustrates these statements : the first user can create a composition for the process of buying a book involving the shipping and online payment services. He can share this composition with the second user to buy things online with goods delivery and payment services. The second user can reuse the same composition if he wants to buy a book, or only a part of this composition if he wants to buy computer accessories. The creation of a new composition as well as its reuse depend on the discovery process discussed below.

4.4 Conclusion

Composition can be defined as the process of invoking collaboratively and combining inputs and outputs of more than one service in order to achieve an end-goal not possible to achieve otherwise with only one service. Despite the evolution of service technologies, the need for service composition to build complex applications is still present because of the distributed nature of the Web. However, the challenges we have to overcome have changed.

The interlinked nature of linked Web services opens more opportunities to discover related services that can potentially contribute to solving the user's request. However, as linked RESTful services rely on uniform interfaces, stateless interactions and loose coupling between client and services, composing linked services is different than in classic SOA services.

Automating the process of composing multiple services' functionality in order to answer to a user's high level request remains an open research problem. The advent of RESTful Web services has changed the way service composition is done because of the REST architectural style constraints that force a uniform interface on services and make the client manage the application state. This means that the server no longer stores the application state, and the composition has to be carried out by the client. Many solutions have been proposed, some of them adapt the classic Web service approaches to the REST architecture. Some of them rely on very specific semantic description format in order to automated the process of composition with the help of discovery and selection.

In this chapter, we show how composition can be achieved with the help of the aforementioned description model and algorithms and we propose a solution that allows users to *store*, *reuse* and *share* flexible and dynamically generated composition workflows/mashups that perform specific complex tasks otherwise impossible to perform with a single service. Services part of a specific stored workflow can be replaced dynamically, if unavailable at the time of the execution for example, without changing the whole process as long as it offers similar functionality and its inputs/outputs are compatible.

In future work, we aim to enable automatic reuse of compositions by reasoning about their semantic annotations in order to respond to a user's request. Composition directories allow clients to answer complex user requests that involve multiple services on the Web. Another interesting perspective that builds upon this, is to allow a context based service composition. In other words, if the user's context has changed, adapt the services used in the composition process according to the new variables. This could prompt the execution of the discovery and selection algorithms in order to account for the changes that happen on the service level as well.

Chapter 5

General Conclusion

The convergence of the Web service technologies and the service orientation with the technologies of the semantic Web, Linked Data and the Web of data is a promising research area. Creating a framework where functionality is provided by services and data is published according to Linked Data practices would be a huge step forward towards achieving true automation of task resolution on the Web.

On the one hand, services, as well-defined, independent and distributed pieces of functionality, are a very powerful tool for developing distributed systems and allowing the processing of data on-demand. On the other hand, Linked Data provides the means to publish interconnected data described with machine-readable semantic annotations that adhere to the principles and practices of the semantic Web.

The advent of the REST architectural style as a dominant paradigm for the design and implementation of services and APIs on the Web has changed the way clients interact with the Web. The classic Web services using the XML stack of technologies such as SOAP, WSDL and UDDI is slowly being abandoned in favour of the RESTful APIs. On the one hand, the properties REST offers are desirable in an open information space like the Web. Scalability of the REST architecture allows it to accommodate an ever increasing amount of data and users, without loss in efficiency. The simplicity and reliability of services and clients attracts more businesses to expose their services on the Web at low costs. On the other hand, however, the constraints imposed by REST require revisiting the mechanisms typically adopted in the RPC-oriented classic Web services regarding many aspects. Of particular interest is the resource-oriented nature of the Web rather than the input-operation-output paradigm adopted in classic Web services. Description of service functionality, interlinking and other properties is the first aspect. Since REST imposes the constraint of a uniform interface, which is done by the HTTP protocol on the Web, the description of operations, and consequently functionality, is done differently

than describing operations offered by a SOAP-based service. The HATEOAS constraint imposes that links in resource representations drive the application state, which is kept on the client side, rather than service side like in classic approaches. Discovery and selection of services is the second aspect. Making functionality, in the form of services, searchable is of utmost importance in order to allow the reuse of distributed software on the Web. Client need to be able to access the Web to locate and filter services based on their functionality, using service descriptions. Composition of services is the third aspect. We can no longer rely on manually made service mashups that are inflexible to the dynamic nature of the Web. Automated clients should be able to discover functionality and the Web and orchestrate, with the help of descriptions, the functionality from different service providers in order to answer complex user requests in a fully automated manner.

5.1 Research Problems

5.1.1 Description

The current solutions for describing RESTful linked Web services are limited. The solutions that worked for classic Web services are considered too complex, verbose and heavyweight for the resource-oriented and lightweight nature of the RESTful APIs. The proposals for the specific description of RESTful services are either still too complex or do not take into account the entirety of the REST constraints, ignoring important concepts such as the hypermedia-driven application state.

This is why there is a need to establish a description model that takes into account all the properties of the REST architecture, while at the same time remain modular, extensible and lightweight. In order to support the automatic discovery, selection and composition of RESTful services, the service description must include explicitly semantic descriptions of the functionality and inputs/outputs of the service, its interlinking with other resources in the Web as well as other descriptive information such as non-functional properties and annotations of data contained in the resources offered by the service. The reuse of vocabularies for the semantic annotation of descriptions is important in order to allow a seamless integration with the Linked Data practices.

5.1.2 Discovery

The centralized solutions to discover classic Web services have suffered from many problems, most important of which was scalability on the Web. In order to make functionality

searchable on the Web, there is a need to describe the operations provided by RESTful services as well as their inputs and outputs. We need also to make services link to each other, while describing the nature of the links in question. This makes the Web of services a huge directed, labelled and sometimes weighted multigraph. Discovery consists in traversing this huge graph to locate services providing the functionality needed to answer a user's request.

5.1.3 Selection

Discovery of functionality on the Web can sometimes give huge number of matches. We need to offer a solution to filter the results of the discovery algorithm and only offer the user the most relevant service for the requested task, while taking into account his preferences and requirements. Selection uses the descriptions of the non-functional properties of the service in order to filter the service matches.

5.1.4 Composition

The composition of service using mashups has a number of limitations, such the inflexibility and the need to remake the mashup if a service is no longer available. With explicitly described services and an automatic discovery of functionality on the Web, the automatic composition of functionality can be achieved. The composition engine in a RESTful architecture is the client, which is responsible for making calls to service operations and integrating the results of the different alls in order to achieve the user's goal.

5.2 Contribution Summary

With the help of a motivating scenario we demonstrated the need for solutions to the problems we presented earlier. We took the example of a user that wishes to buy a book online. All the user has to do is enter a high level request to the client, which has to automatically answer the request by discovering services on the Web that answer each of the tasks required by the request. We used the scenario as a proof of concept illustration for the contributions we summarize in the following.

5.2.1 Descriptors

We propose a description model to semantically annotate different aspects of RESTful services, namely :

— The data in the resource representations.
— The operations provided by the service.
— The links to related resources.
— The non-functional properties of the service.

We group this descriptive information in a single data structure we call a *descriptor*, link each resource representation to its descriptor and annotate the link with explicit semantics.

5.2.2 HATEOAS-based discovery algorithm

We propose a discovery algorithm that discovers services that can provide functionality to answer each of the tasks required by the user's request. The algorithm uses the semantic annotations on the service operations to determine whether or not a service is a candidate to perform the task in question. The algorithm relies on the semantically annotated links between resources in order to traverse the graph of services formed by the links. We opted for a breadth-first approach in order to maximize the result diversity since most of the services include links to related services in their root resource.

5.2.3 On-the-fly selection

We propose a selection algorithm using a skyline-based approach in order to filter the candidates for each task. The algorithm is executed in parallel with the discovery process and takes as input the results being discovered. We detailed multiple configurations of the algorithm in order to make a compromise between performance and result accuracy.

5.2.4 Composition Directories

We propose the concept of *Composition Directories* as an alternative to mashups. Composition directories allow the client to automatically discover new services for certain tasks in order to replace the ones that are no longer available. This allows the emergence of new and enticing service providers as alternatives. They also allow users to share their composition directories with other users in a social-media-like setup.

5.3 Perspectives

5.3.1 Description

We discussed previously how separating representations and descriptions allows an incremental annotation of services with semantics, and allows to describe existing services with minimal effort and cost. As we move forward, and the integration of RESTful linked Web services on the semantic Web becomes more popular and widely spread, we can directly embed the descriptions inside service representations, the same way some of the description proposals do with microformats on HTML code. This offers the advantage of having to maintain only the representation of the resource with the descriptions included, and the retrieval of descriptions can be done at the same time of representation, at the moment of manipulating the resource.

As the practices of Linked Data and the semantic Web become more popular, we noticed the emergence of service wrappers that expose data contained in data silos, which previously operated in private environments, as Linked Data. These interfaces represent Linked Services that expose Linked Data on the Web. However, describing such services in a manner that complies with our proposal for description has not been explored in this thesis. Proposing a process that can automatically generate semantic descriptions for such service wrappers can be an interesting initiative in order to allow a better integration of the newly exposed data on the Semantic Web.

When it comes to service descriptions, user-related feedback and rating can be of utmost importance in deciding whether or not to use a service and its links in answering the user's request. Although the objective of our work is to automate service manipulation tasks, we do not exclude completely the intervention of the human user in guiding these tasks. In fact, human intervention can be sometimes desired by the user in order to steer the machine client in the right direction when it comes to answering his request. Integrating such dynamic descriptive information onto descriptors and exploiting it to refine the discovery and selection processes opens new interesting challenges. Also, the idea of semantically annotating such descriptive information related to user rating and feedback allows the exploitation of the semantic Web technologies and Linked Data principles in order to automate the generation of these descriptions and embedding them to resource representations.

5.3.2 Discovery

We used a very basic variant of the breadth first search algorithm for our discovery solution in order to illustrate how semantic descriptions enable a semantically guided automatic discovery process. Using advanced heuristics can help reach a better compromise between performance and result diversity. Exploring links based on a dynamically calculated similarity, between the current resource and the target resource on one hand, and the similarity between the desired concept and the concepts describing operations on the target resource on the other hand, can help increasing consistency of the results while saving time by not exploring entire branches not relevant to the request.

Also, we need to address the issue of allowing service providers link their services with other complementary services on the business and data level. This can be achieved by implementing automated agents that crawl through the descriptions of services on the Web and create links based on the semantics of the functionality and data provided by services as well as business defined criteria (partnerships for example).

5.3.3 Selection

We used a minimal quality of service model in order to illustrate our selection algorithm. The model we used does not reflect how real business services evaluate their quality, price, availability, etc. An interesting perspective would be to integrate a complete quality of service model to the descriptions we provide to services and resources. The selection algorithm's consistency would be vastly enhanced.

Also, capturing the user's preferences can be expanded in order to include more context-based information related to the user's browsing history, location, language, time of the day, time of the year, the device being used and many more elements that can help the selection process to automatically make the decision of using a certain service instead of the other.

5.3.4 Composition

Composition directories allow clients to answer complex user requests that involve multiple services on the Web. An interesting perspective that builds upon this, is to allow a context based service composition. In other words, if the user's context has changed, adapt the services used in the composition process according to the new variables. This could prompt the execution of the discovery and selection algorithms for another iteration in order to account for the changes that happen on the service level as well, or prompt

the exploration of linked composition directories in order to assess whether or not some of the new trending services used by the other users can be used to obtain better results.

Also, we do not treat in our work data conversion between incompatible services. This idea allows the interoperation of initially incompatible services by introducing intermediary services that can act as data converters in order to transform the outputs of a service into usable inputs for the next service despite them being incompatible.

5.3.5 Other Perspectives

Another interesting perspective for future work is handling data concerns in RESTful linked service environments. Important questions such as :

— how to ensure the quality of data stored and exchanged by RESTful linked services ?
— how to make sure the data provided by RESTful linked services is reliable ?
— how to bestow protection and handle security problems when it comes to client-service exchanges ?

arise and need to be answered in order to enable a solid and robust service-oriented environment on the semantic Web.

Bibliography

[Alarcon and Wilde, 2010] Alarcon, R. and Wilde, E. (2010). Linking data from restful services. In *Third workshop on linked data on the web, raleigh, north carolina (april 2010)*.

[Alarcón et al., 2010] Alarcón, R., Wilde, E., and Bellido, J. (2010). Hypermedia-Driven RESTful Service Composition. In Maximilien, E. M., Rossi, G., Yuan, S.-T., Ludwig, H., and Fantinato, M., editors, *ICSOC Workshops*, volume 6568 of *Lecture Notes in Computer Science*, pages 111–120.

[Alrifai et al., 2010] Alrifai, M., Skoutas, D., and Risse, T. (2010). Selecting skyline services for qos-based web service composition. In *Proceedings of the 19th international conference on World wide web*, pages 11–20. ACM.

[Anadiotis et al., 2009a] Anadiotis, G., Kotoulas, S., Lausen, H., and Siebes, R. (2009a). Massively scalable web service discovery. In Awan, I., Younas, M., Hara, T., and Durresi, A., editors, *The IEEE 23rd International Conference on Advanced Information Networking and Applications, AINA 2009, Bradford, United Kingdom, May 26-29, 2009*, pages 394–402. IEEE Computer Society.

[Anadiotis et al., 2009b] Anadiotis, G., Kotoulas, S., Lausen, H., and Siebes, R. (2009b). Massively scalable web service discovery. *2009 International Conference on Advanced Information Networking and Applications*, pages 394–402.

[Anadiotis et al., 2009c] Anadiotis, G., Kotoulas, S., Lausen, H., and Siebes, R. (2009c). Massively scalable web service discovery. In *2009 International Conference on Advanced Information Networking and Applications*, pages 394–402. IEEE.

[Archer et al., 2009] Archer, P., Smith, K., and Perego, A. (2009). Protocol for web description resources (powder) : Description resources. *W3C Working Draft (April 2009) http ://www. w3. org/TR/powder-dr*.

[Bennara et al., 2015a] Bennara, M., Amghar, Y., and Mrissa, M. (2015a). Managing web resource compositions. In *2015 IEEE 24th International Conference on Enabling Technologies : Infrastructure for Collaborative Enterprises*, pages 176–181. IEEE.

[Bennara et al., 2015b] Bennara, M., Amghar, Y., and Mrissa, M. (2015b). Managing web resource compositions. In Reddy, S., editor, *24th IEEE International Conference on Enabling Technologies : Infrastructure for Collaborative Enterprises, WETICE Workshops 2015, Larnaca, Cyprus, June 15-17, 2015*, pages 176–181. IEEE.

[Bennara et al., 2014a] Bennara, M., Mrissa, M., and Amghar, Y. (2014a). An approach for composing restful linked services on the web. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 977–982. ACM.

[Bennara et al., 2014b] Bennara, M., Mrissa, M., and Amghar, Y. (2014b). An approach for composing restful linked services on the web. In Chung, C., Broder, A. Z., Shim, K., and Suel, T., editors, *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume*, pages 977–982. ACM.

[Bennara et al., 2016a] Bennara, M., Mrissa, M., and Amghar, Y. (2016a). Linked service selection using the skyline algorithm. In *International Conference on Model and Data Engineering*, pages 88–97. Springer.

[Bennara et al., 2016b] Bennara, M., Mrissa, M., and Amghar, Y. (2016b). Semantic-enabled and hypermedia-driven linked service discovery. In *International Conference on Model and Data Engineering*, pages 108–117. Springer.

[Berners-Lee, 1998] Berners-Lee, T. (1998). Notation3. *http ://www. w3. org/DesignIssues/Notation3. html*.

[Berners-Lee et al., 1994] Berners-Lee, T., Dimitroyannis, D., Mallinckrodt, A. J., and McKay, S. (1994). World wide web. *Computers in Physics*, 8(3) :298–299.

[Berners-Lee and Fischetti, 2000] Berners-Lee, T. and Fischetti, M. (2000). *Weaving the Web : the past, present and future of the World Wide Web*. Texere.

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

[Bizer et al., 2009b] Bizer, C., Heath, T., and Berners-Lee, T. (2009b). Linked data-the story so far. *Semantic Services, Interoperability and Web Applications : Emerging Concepts*, pages 205–227.

[Borzsony et al., 2001] Borzsony, S., Kossmann, D., and Stocker, K. (2001). The skyline operator. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, pages 421–430. IEEE.

[Bournez, 2005] Bournez, C. (2005). Team comment on web service modeling ontology (wsmo) submission. *W3C Submissions*.

[Bülthoff and Maleshkova, 2014] Bülthoff, F. and Maleshkova, M. (2014). Restful or restless - current state of today's top web apis. In Presutti, V., Blomqvist, E., Troncy,

R., Sack, H., Papadakis, I., and Tordai, A., editors, *The Semantic Web : ESWC 2014 Satellite Events - ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*, volume 8798 of *Lecture Notes in Computer Science*, pages 64–74. Springer.

[Chinnici et al., 2007] Chinnici, R., Moreau, J.-J., Ryman, A., and Weerawarana, S. (2007). Web services description language (wsdl) version 2.0 part 1 : Core language. *W3C recommendation*, 26(1) :19.

[Christensen et al., 2001] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., et al. (2001). Web services description language (wsdl) 1.1.

[Clement et al., 2004] Clement, L., Hately, A., Riegen, C., and Rogers, T. (2004). Universal description discovery & integration (uddi) 3.0. 2. *Organization for the Advancement of Structured Information Standards (OASIS). Specification.*

[Dijkstra, 1982] Dijkstra, E. W. (1982). On the role of scientific thought. In *Selected writings on computing : a personal perspective*, pages 60–66. Springer.

[Domingue et al., 2011] Domingue, J., Pedrinaci, C., Maleshkova, M., Norton, B., and Krummenacher, R. (2011). Fostering a relationship between linked data and the internet of services. In Domingue, J., Galis, A., Gavras, A., Zahariadis, T. B., Lambert, D., Cleary, F., Daras, P., Krco, S., Müller, H., Li, M., Schaffers, H., Lotz, V., Alvarez, F., Stiller, B., Karnouskos, S., Avessta, S., and Nilsson, M., editors, *The Future Internet - Future Internet Assembly 2011 : Achievements and Technological Promises*, volume 6656 of *Lecture Notes in Computer Science*, pages 351–366. Springer.

[Fielding, 2000] Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures.* PhD thesis, University of California, Irvine. AAI9980887.

[Förster and Wattenhofer, 2012] Förster, K.-T. and Wattenhofer, R. (2012). Directed graph exploration. In *Principles of Distributed Systems*, pages 151–165. Springer.

[Ge and Qiu, 2008] Ge, J. and Qiu, Y. (2008). Concept similarity matching based on semantic distance. In *Semantics, Knowledge and Grid, 2008. SKG'08. Fourth International Conference on*, pages 380–383. IEEE.

[Hadley, 2006] Hadley, M. J. (2006). Web application description language (wadl). *W3C member submission.*

[Hansen et al., 2002] Hansen, M., Madnick, S., and Siegel, M. (2002). Data integration using web services. In *Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web*, pages 165–182. Springer.

[Hatzi et al., 2012] Hatzi, O., Batistatos, G., Nikolaidou, M., and Anagnostopoulos, D. (2012). A specialized search engine for web service discovery. In *2012 IEEE 19th International Conference on Web Services*, pages 448–455. IEEE.

[Hau et al., 2005] Hau, J., Lee, W., and Darlington, J. (2005). A semantic similarity measure for semantic web services. In *Web Service Semantics Workshop at WWW*, pages 10–14.

[John and Rajasree, 2013] John, D. and Rajasree, M. S. (2013). RESTDoc : Describe, Discover and Compose RESTful Semantic Web Services using Annotated Documentations. *International Journal of Web & Semantic Technology (IJWesT)*, 4(1).

[Kopecký et al., 2008] Kopecký, J., Gomadam, K., and Vitvar, T. (2008). hrests : An html microformat for describing restful web services. *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 1 :619–625.

[Kopeckỳ et al., 2008] Kopeckỳ, J., Gomadam, K., and Vitvar, T. (2008). hrests : An html microformat for describing restful web services. In *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, volume 1, pages 619–625. IEEE.

[Kopecky and Vitvar, 2008] Kopecky, J. and Vitvar, T. (2008). microwsmo : Semantic description of restful services (wsmo working draft). Technical report, Technical report, WSMO, 2008. http ://www.wsmo.org/TR/d38/v0.1.

[Kopeckỳ et al., 2007] Kopeckỳ, J., Vitvar, T., Bournez, C., and Farrell, J. (2007). Sawsdl : Semantic annotations for wsdl and xml schema. *IEEE Internet Computing*, 11(6) :60–67.

[Kovatsch et al., 2015] Kovatsch, M., Hassan, Y. N., and Mayer, S. (2015). Practical semantics for the internet of things : Physical states, device mashups, and open questions. In *Internet of Things (IOT), 2015 5th International Conference on the*, pages 54–61. IEEE.

[Kozen, 1992] Kozen, D. (1992). Depth-first and breadth-first search. In *The Design and Analysis of Algorithms*, Texts and Monographs in Computer Science, pages 19–24. Springer New York.

[Lanthaler et al., 2010] Lanthaler, M., Granitzer, M., and Gütl, C. (2010). Semantic web services : state of the art. In *Proceedings of the IADIS international conference-Internet technologies and society 2010*, pages 107–114. IADIS Press.

[Lanthaler and Guetl, 2013] Lanthaler, M. and Guetl, C. (2013). Hydra : A Vocabulary for Hypermedia-Driven Web APIs. In Bizer, C., Heath, T., Berners-Lee, T., Hausenblas, M., and Auer, S., editors, *LDOW*, volume 996 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Lanthaler and Gütl, 2010] Lanthaler, M. and Gütl, C. (2010). Towards a restful service ecosystem. In *4th IEEE International Conference on Digital Ecosystems and Technologies*, pages 209–214. IEEE.

[Lassila and Swick, 1999] Lassila, O. and Swick, R. R. (1999). Resource description framework (rdf) : Model and syntax specification. Recommendation, World Wide Web Consortium. http://www.w3.org/TR/REC-rdf-syntax/.

[Lathem et al., 2007] Lathem, J., Gomadam, K., and Sheth, A. P. (2007). Sa-rest and (s) mashups : Adding semantics to restful services. In *International conference on semantic computing (ICSC 2007)*, pages 469–476. IEEE.

[Maamar et al., 2011a] Maamar, Z., Wives, L. K., Badr, Y., Elnaffar, S., Boukadi, K., and Faci, N. (2011a). Linkedws : A novel web services discovery model based on the metaphor of social networks. *Simulation Modelling Practice and Theory*, 19(1) :121–132.

[Maamar et al., 2011b] Maamar, Z., Wives, L. K., Badr, Y., Elnaffar, S., Boukadi, K., and Faci, N. (2011b). Linkedws : A novel web services discovery model based on the metaphor of "social networks". *Simulation Modelling Practice and Theory*, 19(1) :121–132.

[Maleshkova et al., 2009] Maleshkova, M., Kopecký, J., and Pedrinaci, C. (2009). Adapting sawsdl for semantic annotations of restful services. In *OTM Workshops*.

[Martin et al., 2004] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., et al. (2004). Owl-s : Semantic markup for web services. *W3C member submission*, 22(4).

[M'Barek and Tata, 2008] M'Barek, N. O. A. and Tata, S. (2008). Services web : revue des approches de description sémantique. In *SIIE 2008 : Système d'Information et Intelligence Economique*, pages 1–7.

[Najork and Wiener, 2001] Najork, M. and Wiener, J. L. (2001). Breadth-first crawling yields high-quality pages. In *Proceedings of the 10th international conference on World Wide Web*, pages 114–118. ACM.

[Pautasso, 2009a] Pautasso, C. (2009a). Restful web service composition with bpel for rest. *Data Knowl. Eng.*, 68(9) :851–866.

[Pautasso, 2009b] Pautasso, C. (2009b). Restful web service composition with bpel for rest. *Data Knowl. Eng.*, 68 :851–866.

[Pedrinaci et al., 2014a] Pedrinaci, C., Cardoso, J., and Leidig, T. (2014a). Linked usdl : a vocabulary for web-scale service trading. In *European Semantic Web Conference*, pages 68–82. Springer.

[Pedrinaci et al., 2014b] Pedrinaci, C., Cardoso, J., and Leidig, T. (2014b). Linked usdl : A vocabulary for web-scale service trading. In Presutti, V., d'Amato, C., Gandon, F., d'Aquin, M., Staab, S., and Tordai, A., editors, *ESWC*, volume 8465 of *Lecture Notes in Computer Science*, pages 68–82. Springer.

[Pedrinaci et al., 2010a] Pedrinaci, C., Domingue, J., et al. (2010a). Toward the next wave of services : Linked services for the web of data. *J. ucs*, 16(13) :1694–1719.

[Pedrinaci et al., 2010b] Pedrinaci, C., Domingue, J., and Krummenacher, R. (2010b). Services and the web of data : An unexploited symbiosis. In *2010 AAAI Spring Symposium Series*.

[Prud'hommeaux and Seaborne, 2008] Prud'hommeaux, E. and Seaborne, A. (2008). Sparql query language for rdf. recommendation, w3c.

[Ran, 2003] Ran, S. (2003). A model for web services discovery with qos. *SIGecom Exchanges*, 4(1) :1–10.

[Roman et al., 2005] Roman, D., Keller, U., Lausen, H., De Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., and Fensel, D. (2005). Web service modeling ontology. *Applied ontology*, 1(1) :77–106.

[Russell et al., 1995] Russell, S., Norvig, P., and Intelligence, A. (1995). A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25.

[Schreiber and Dean, 2004] Schreiber, G. and Dean, M. (2004). Owl web ontology language reference. `http://www.w3.org/TR/2004/REC-owl-ref-20040210/`.

[Sharifi and Bayram, 2016] Sharifi, O. and Bayram, Z. (2016). A critical evaluation of web service modeling ontology and web service modeling language. In *International Symposium on Computer and Information Sciences*, pages 97–105. Springer.

[Sporny et al., 2014] Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., and Lindström, N. (2014). Json-ld 1.0. *W3C Recommendation*, 16 :41.

[Stadtmüller, 2012] Stadtmüller, S. (2012). Composition of linked data-based restful services. In *International Semantic Web Conference*, pages 461–464. Springer.

[Verborgh et al., 2013] Verborgh, R., Hausenblas, M., Steiner, T., Mannens, E., and de Walle, R. V. (2013). Distributed affordance : an open-world assumption for hypermedia. In Carr, L., Laender, A. H. F., Lóscio, B. F., King, I., Fontoura, M., Vrandecic, D., Aroyo, L., de Oliveira, J. P. M., Lima, F., and Wilde, E., editors, *WWW (Companion Volume)*, pages 1399–1406. International World Wide Web Conferences Steering Committee / ACM.

[Verborgh et al., 2011a] Verborgh, R., Steiner, T., Deursen, D. V., Roo, J. D., de Walle, R. V., and Vallés, J. G. (2011a). Description and Interaction of RESTful Services for Automatic Discovery and Execution. In *Proceedings of the FTRA 2011 International Workshop on Advanced Future Multimedia Services*.

[Verborgh et al., 2012] Verborgh, R., Steiner, T., Van Deursen, D., Coppens, S., Vallés, J. G., and Van de Walle, R. (2012). Functional descriptions as the bridge between hy-

permedia apis and the semantic web. In *Proceedings of the third international workshop on restful design*, pages 33–40. ACM.

[Verborgh et al., 2011b] Verborgh, R., Steiner, T., Van Deursen, D., De Roo, J., Van de Walle, R., and Vallés, J. G. (2011b). Description and interaction of restful services for automatic discovery and execution. In *2011 FTRA International workshop on Advanced Future Multimedia Services (AFMS 2011)*. Future Technology Research Association International (FTRA).

[Vitvar et al., 2008] Vitvar, T., Kopecký, J., Viskova, J., and Fensel, D. (2008). Wsmo-lite annotations for web services. In *ESWC*.

[Wang et al., 2011] Wang, Z.-J., Liu, Z.-Z., Zhou, X.-F., and Lou, Y.-S. (2011). An approach for composite web service selection based on dgqos. *The International Journal of Advanced Manufacturing Technology*, 56(9-12) :1167–1179.

[Wilde, 2010a] Wilde, E. (2010a). Linked data and service orientation. In *International Conference on Service-Oriented Computing*, pages 61–76. Springer.

[Wilde, 2010b] Wilde, E. (2010b). Linked data and service orientation. In Maglio, P. P., Weske, M., Yang, J., and Fantinato, M., editors, *Service-Oriented Computing - 8th International Conference, ICSOC 2010, San Francisco, CA, USA, December 7-10, 2010. Proceedings*, volume 6470 of *Lecture Notes in Computer Science*, pages 61–76.

[Wolsey and Nemhauser, 2014] Wolsey, L. A. and Nemhauser, G. L. (2014). *Integer and combinatorial optimization*. John Wiley & Sons.

[Wong and Gibbons, 2008] Wong, P. Y. H. and Gibbons, J. (2008). A process semantics for bpmn. In Liu, S., Maibaum, T. S. E., and Araki, K., editors, *ICFEM*, volume 5256 of *Lecture Notes in Computer Science*, pages 355–374. Springer.

[Yu et al., 2007] Yu, T., Zhang, Y., and Lin, K. (2007). Efficient algorithms for web services selection with end-to-end qos constraints. *TWEB*, 1(1).

[Zeng et al., 2004] Zeng, L., Benatallah, B., Ngu, A. H. H., Dumas, M., Kalagnanam, J., and Chang, H. (2004). Qos-aware middleware for web services composition. *IEEE Trans. Software Eng.*, 30(5) :311–327.

[Zhao and Doshi, 2009] Zhao, H. and Doshi, P. (2009). Towards automated restful web service composition. *2009 IEEE International Conference on Web Services*, pages 189–196.

[Zhao et al., 2014] Zhao, X., Wen, Z., and Li, X. (2014). Qos-aware web service selection with negative selection algorithm. *Knowledge and Information Systems*, 40(2) :349–373.

FOLIO ADMINISTRATIF

THESE DE L'UNIVERSITE DE LYON OPEREE AU SEIN DE L'INSA LYON

Nom : **BENNARA** Date de soutenance : **18/07/2019**

Prénoms : **Mahdi**

Titre **: Linked Service Integration on the Semantic Web**

Nature : **Doctorat** Numéro d'ordre **: 2019LYSEI0466**

École doctorale : **Informatique et Mathématiques de Lyon**

Spécialité : **Informatique**

Résumé :

L'informatique orientée services facilite l'interopérabilité entre les systèmes distribues. Depuis quelques années, l'émergence du Web sémantique a posé de nouveaux défis pour la communauté de recherche dans les calculs et la compatibilité sémantique des données.
L'approche «services» et le Web sémantique constituent une piste prometteuse pour remédier aux problèmes qui entravent les deux domaines. D'une part l'orientation services permet d'assurer l'interopérabilité des données et des traitements au niveau sémantique, et d'autre part le Web sémantique permet d'automatiser les taches de manipulation de services à un haut niveau.
Dans le cadre de notre travail de recherche, nous avons détaillé les défis que rencontre la communauté de chercheurs dans l'intégration des pratiques de l'orientation services dans le Web sémantique, et plus particulièrement l'intégration des services REST dans l'implémentation du Web qui repose sur les principes du «Linked Data» pour constituer ce que l'on appelle les «RESTful Linked Services». Les défis en question sont : La description, la découverte, la sélection et la composition.
Nous avons proposé une solution pour chacun de ces défis. Les contributions que nous avons proposées sont : la structure de descripteur, un algorithme de découverte sémantique, un algorithme de sélection base sur Skyline et les répertoires de composition.
Nous pensons que l'ensemble de contributions que nous avons proposées peut être adopté par les fournisseurs de services sur le Web afin de faciliter l'intégration des pratiques du sémantique Web avec les technologies des services et de REST en particulier. Ceci permettra donc d'automatiser les taches de manipulation de services a un haut niveau, tel que la découverte sur la base de concepts sémantiques, la sélection sur la base de propriétés non-fonctionnelles et de qualité de services et la composition de plusieurs services hétérogènes, sur le plan des données ainsi que sur le plan des traitements, afin d'obtenir des services composites avec de la valeur ajoutée.

Mots-Clés : **Services Web, Web sémantique, Web des données**

Laboratoire de recherche : **Laboratoire d'InfoRmatique en Image et Systèmes d'information (LIRIS) - UMR 5205**

Directeurs de thèse: **Youssef AMGHAR** & **Michaël MRISSA**

Président de jury :

Composition du jury :

MURISASCO, Elisabeth	**Professeur Univ. de Toulon**	**Rapporteure**
FRONT, Agnès	**MCF–HDR Univ. de Grenoble**	**Rapporteure**
MARET, Pierre	**Professeur Univ. de Saint-Étienne**	**Examinateur**
SAVONNET, Marinette	**MCF–HDR Univ. de Bourgogne**	**Examinatrice**
AMGHAR, Youssef	**Professeur Univ. de Lyon**	**Directeur de thèse**
MRISSA, Michaël	**Professeur Univ. de Pau**	**Codirecteur de thèse**