A Resource-Oriented Architecture for Integration and Exploitation of Linked Data

Pierre de Vettor

Université Claude Bernard Lyon I
UFR Informatique

École doctorale Informatique et
Mathématiques de Lyon

A Resource-Oriented Architecture for Integration and Exploitation of Linked Data

THÈSE

présentée et soutenue publiquement le 29 Septembre 2016

pour l'obtention du

Doctorat de l'Université Claude Bernard Lyon I

(spécialité informatique)

par

Pierre De Vettor

Composition du jury

Rapporteurs : Abderrafia Koukam, Professeur, Université de Technologie de Belfort-Montbéliard,
Khalil Drira, Directeur de Recherche, LAAS-CNRS Toulouse,

Examinateurs : Philippe Lalanda, Professeur, Université Joseph Fourier de Grenoble,
Franck Morvan, Professeur, Institut de Recherche en Informatique de Toulouse,
Marinette Savonnet, Maître de conférence, Université de Bourgogne,

Invités : Salim Berbar, Directeur général et administrateur, Audience Labs, Lyon

Directeurs : Michaël Mrissa, Professeur, Université de Pau et des pays de l'Adour,
Djamal Benslimane, Professeur, Université Claude Bernard Lyon 1,

Laboratoire d'InfoRmatique en Images et Systèmes d'information — UMR 5205

Acknowledgments

First of all, I would like to express my sincere gratitude to my supervisors, for their enthousiasm and great guidance: Michael Mrissa, for his availability, his sympathy, and for the numerous hours he spent in helping, guiding, and motivating me. Djamal Benslimane, for his trust and sympathy. His positive and motivating advice were always a real help throughout this work. I thank them for their continuous encouragement and caring during this thesis. It has been a pleasure working with them.

I also would like to address my best regards to Koukam Abderrafia and Khalil Drira who accepted to review this document. I also extend my thanks to the examiners and jury members Marinette Savonnet, Philippe Lalanda and Franck Morvan.

In a second time, this work would never have been possible without the input of the Audience Labs company, and I would like to express my particular thanks to:

— Salim Berbar for helping and encouraging me into this work, for his availability despite the numerous task that he is responsible for

— Nadir Habrih, Hakim Labiod, Farid Amiour and Eric Gharbi for welcoming me in their company, and for their trust in my work

— my colleagues and friends: Amine Zayed and Soungalo Sidibé for the moments we shared together

I would also like to say thank you to my former lab partners, Rafiqul Haque and Samir Amir who welcomed me in their office room at the beginning, and for the stimulating talks about research.

Thanks to Ruthan Semanou and Marion Normand who took the time to proofread this document.

Many thanks to my parents, sisters, and in-laws for their caring support. A very special thanks to my dear friends: Damien, Paul, Romain, Mélanie, Boris, Clara and Michael.

Finally, I want to thank my wife Anne-Sophie, for her kindness and her unconditional love, and for encouraging me all this time. Thank you.

To my wife Anne-Sophie and my daughter Charlotte,

Résumé

Cette thèse porte sur l'intégration de données provenant de sources hétérogènes sur le Web. L'objectif est de fournir une architecture générique et modulable capable de combiner, de façon intelligente, ces données hétérogènes dans le but de les rendre réutilisables. Ce travail est motivé par le scenario de l'entreprise Audience Labs qui fournira les sources de données permettant la mise à l'échelle de cette architecture.

Dans ce rapport, nous proposons une approche permettant de s'adapter à la diversité des sources de données, avec des problématiques de gestion dynamique des sources, de passage à l'échelle et de consistance des données produites (cohérentes, sans erreurs, ni doublons).

Pour répondre à ces problématiques, nous proposons un méta-modèle pour représenter ces sources selon leurs caractéristiques, liées à l'accès (URI), à l'extraction (format), mais aussi aux capacités physiques de ces sources (latence, volume). En nous appuyant sur cette formalisation, nous proposons différentes stratégies pour adapter les traitements aux spécificités.

Basé sur ces modèles, nous proposons une architecture RESTful où tous les composants sont accessibles à travers HTTP via leurs URIs. En nous basant sur les caractéristiques des sources, nous pouvons alors générer des workflows d'exécution spécifiques et adaptés. Ils permettent d'orchestrer les différentes tâches du processus d'intégration de façon optimale, en donnant différentes priorités à chacune des tâches. Ainsi, les temps de traitement sont diminués, ainsi que les volumes des données échangées.

Afin d'améliorer la qualité des données produites par notre approche, l'accent est mis sur l'incertitude qui peut apparaître dans les données sur le Web. Nous proposons un modèle de description de l'incertitude à travers le concept de ressource Web incertaine. Celui ci est basé sur un modèle probabiliste où chaque ressource peut avoir plusieurs représentations possibles, avec une certaine probabilité. En nous basant sur ce modèle, nous proposons une approche permettant d'évaluer des requêtes dans un contexte incertain.

Mots-clés: architecture orientée ressource, adaptation, smart data, incertitude, intégration de données, sémantique

Abstract

In this thesis, we focus on data integration of raw data coming from heterogeneous and multi-origin data sources on the Web. In this context, we focus on a specific part of data integration, which is the combination of data sources. The objective is to provide a generic architecture able to analyze and combine this heterogeneous, informal, and sometimes meaningless data into a coherent smart data set. We define smart data as significant, semantically explicit data, ready to be used to fulfill the stakeholders' objective. This work is motivated by a live scenario from the French *Audience Labs* company.

In this report, we propose new models and techniques to adapt the combination process to the diversity of data sources. We focus on transparency and dynamicity in data source management, scalability and responsivity according to the number of data sources, adaptability to data source characteristics, and finally consistency of produced data (coherent data, without errors and duplicates).

In order to address these challenges, we first propose a meta-model in order to represent the variety of data source characteristics, as a set of functional (URI, auth, format), or non-functional properties (volume, latency). By relying on this formalization of data sources, we define different strategies in order to adapt access and processing to data source capabilities.

With help from these models and strategies, we propose a resource-oriented architecture, where each component is accessible through REST via its URI. The orchestration of the different tasks of the combination process can be done in an optimized way, regarding data sources and characteristics. We rely on these characteristics to generate adapted workflow, where tasks will be executed in a specific order that will help to optimize the quantity of data to process, and will reduce execution time.

In order to improve the quality of our approach, we then focus on the data uncertainty that could appear in a Web context, and propose a model to represent this uncertainty. We introduce the concept of uncertain Web resource, based on a probabilistic model where each resource can have different possible representations, each with a probability. This approach will be the basis of a new architecture optimization allowing to take uncertainty into account during our combination process.

Keywords: resource-oriented architecture, workflow, adaptation, smart data, data uncertainty, data integration, data semantics

Contents

List of Figures

Chapter 1

Introduction

Nowadays, with the emerging presence of social media and networking systems, users adapt their online *way of life* and become both data providers and consumers. Every day, huge quantities of data are produced and refined by millions of users, for different purposes. First of all, users can gather data in order to collect and share knowledge, such as culture and history, as shown by the numerous existing archive repositories such as https://archive.org and https://data-archive.ac.uk, where users and organizations have made thousands of ancient documents available. The data can also be used in order to organize or produce exhaustive knowledge about subjects, like those on the https://www.wikipedia.org/ platform and other similar encyclopedia projects. Every day, millions of users voluntarily contribute to thousands of subjects such as sports, science, and politics. More recently, data has also served an educational purpose, as proved by the success of numerous massive open online course (MOOC) platforms. Secondary, data can also be collected online to create profiles, which are online representations of users' personalities. These profiles are built up on several types of platforms, like social networks, like https://myspace.com which had a huge success in its time, followed closely by https://facebook.com, now joined by 1.65 billions of users (March 2016). On these platforms, users recreate their friends networks, manually by indicating their type of relationship, and listing their areas of interest, which are then used by advertising companies to display targeted ads. Users can also decide to make a list their interests for different purposes, such as making a list of things they desire (amazon wishlist) or to benefit from recommendation systems. In this last category, we clearly identify two types of platforms. On the one hand, platforms that automatically gather statistics like https://last.fm which provides similar recommended artists according to the user's playback statistics, and http:

//netflix.com that advises users on the most relevant movies to watch according to the user's watched history. On the other hand are the platforms where users provide grades and evaluations of their favorite products and of the products they dislike, in order to obtain the same kind of recommendations (systems such as http://imdb.com and http://vodkaster.com/ for movies).

These new ways of producing and using data create a new data paradigm, forcing organizations and companies to adopt new data-driven strategies. Thanks to initiatives such as the open data project [Heath and Bizer, 2011], governments and companies have also opened their data to the world across the Web. Data is published and accessible as tabular files (CSV) or via Web APIs [Maleshkova et al., 2010] or SPARQL endpoints. It has lead them to open and improve their information systems, adding more structure and semantics, and drawing benefits from the aggregation of data from the Web. This freely available data can be combined in service mashups [Benslimane et al., 2008], or processed on-the-fly through low-cost and queryable Linked Data endpoints [Verborgh et al., 2016] to produce highly valuable services. For example, the sets of APIs provided by Twitter, Facebook, LinkedIn, Amazon, Youtube, Flickr or Dropbox are reused in thousands of mashups (see also http://www.programmableweb.com/). All these specific business mashups enrich data with semantics and combine several data sets to improve data reuse, providing advanced statistics and useful data for decision support systems.

As exciting as it is, automating the combination of publicly available data from different sources, still remains a complex task, and there is a lack of a generalized approach. Current existing mashups are usually created for specific purposes, such as answering to answer specific queries about specific data sources. For example, we could imagine a company which has an innovative idea for a mobile application to manage trips and reserve plane and train tickets, directly connected to hotels in the desired cities. Typically, they construct a specific workflow using the different available APIs to compute an itinerary, look at available tickets, and request hotel rooms with location and prices. In the end, they will connect each of these different APIs by hard coding interactions in the application.

There is need for a system that could, depending on a specific use case, retrieve the relevant data sources and data, analyze the provided data (format, concepts, etc.), and design a suitable workflow for this use case, all this with a minimal user input. It requires a high level of adaptation in order to automatically access data sources and extract their data, since information systems do not always respect the same standards,

formats or schemas. Many heterogeneities related to differences in structure, syntax or semantics may exist when trying to handle these data sources. Each data source can have different characteristics, using different protocols, and handling different formats. Different types of data sources may also require specific processing which can constrain their usage, like an authentication protocol or quotas limiting access. Data sources can also be affected by physical properties, such as network latency, high volume, and update frequency. Finally, data sources are sometimes subject to uncertainty, meaning that the data they contain can be contradictory to another data source, or they can be untrustworthy, due to failure in collection mechanisms. There is a need for a solution, in order to take all this information into account, and to adapt the process to each data source. Doing so, we could propose a transparent solution, which would be able to process any kind of data sources, and since new technologies are proposed every day, there is a need for a system which can quickly adapt to new types of data source characteristics.

The objective of this work is to propose a solution to automatically combine multi-origin and heterogeneous data sources in the form of a resource-oriented architecture, which would take better advantage of publicly available data. This system should be able, according to a query and a set of data sources, to select the relevant data sources, analyze their needs and generate an adapted workflow that would extract data and combine them. In parallel, our goal is to upgrade an existing enclosed information system from a company into an open-to-the-world system, in which it will be able to integrate freely available data, according to the different needs of the market. The aim is to propose an adaptive solution for combining multi-origin data sets available from internal information systems and from the Web, in order to produce significant, semantically explicit data, ready to be used to fulfill the stakeholders' objectives. We call this significant and semantically explicit data set *Smart Data*.

1.1 Motivating scenario

In the context of our work, our approach is motivated by a real world scenario from the Audience Labs company, and we focus on the enrichment and reusability of data handled by this company. Data reuse is the process that helps to gain benefits from data that has been stored in various types of data sources. Audience Labs is a French company whose main business is digital marketing, which they practise through the broadcast of advertisement campaigns and newsletters. Audience Labs has a need for

a system that is able to extract data from several non-connected internal data sources, and to confront this data with the external world. The company provides different use cases from automatic statistic reports about previous broadcasts to decision help and recommendation systems, in order to improve future broadcast impact.

Campaign broadcasting consists of sending documents such as newsletters or advertisement campaigns to a set of customers. Typically, these documents are static email-embedded HTML pages containing texts, images and links. We describe each of these documents' contents with a set of concepts which we call interests, related to content keywords and described in ontologies such as DBPedia [1]. Broadcasted documents include transparent tracking mechanisms that allow us to harvest contextual data about users' interactions. Contextual data about the customer wich is saved in our customer profile database for later use, includes browser information, action date and time, and IP addresses. This contextual information, associated with the description and the keywords of the documents can help us to create and extend a profile for each customer.

The scenario objectives, in terms of use cases applied to this global scenario, are :

— studying the impact of a broadcasting campaign on a set of customers (i.e. extracting interesting information and statistics about campaign broadcasts, users' habits and points of interests)

— automatically combining all the available and relevant data sources in order to build a profile for each customer

— providing a help decision tool that will help to choose and extract a set of customers for future broadcasts

On top of that, it is important to make the solution as generic as possible, in order to be able to adapt to future unplanned and/or punctual demands (query interface).

Use case example

A classical use case of the scenario describes the following data sources, each of them presenting different characteristics.

1. an internal linked service giving access to our company's business data
2. an SQL database containing a large volume of information (around 100Gb)
3. a database that records user activities (high volume of changing data) with 10.000/20.000 new tuples per day

1. http://dbpedia.org/

4. a stream of update requests
5. external APIs (twitter, facebook, dbpedia, etc.)

Each of these data sources presents several characteristics. These characteristics are specific to the scenario.

Source 1 is a linked service, i.e. it consumes and produces linked data. It provides access to a small data set that describes the business data of the company. Data pieces that come from this source are subject to privacy constraints. **Source 2** is an SQL endpoint to a database that contains millions of tuples with no semantic annotations. This data source has a low update frequency. **Source 3** is an SQL endpoint to a database that contains users' daily activities. Data from this source are updated regularly, so it requires freshness. **Source 4** is an RSS stream that contains user update requests. It mostly contains data which has to be saved or removed from data results (blacklist information). Other **sources** are represented by a set of APIs (Twitter, Facebook) that help construct interest profiles, as well as a Dbpedia SPARQL endpoint for concept manipulation.

The use case is based on the following interactions:
— starting from a new campaign that has to be broadcasted
— extracting information about this campaign (points of interests, attached concepts, etc.)
— looking at similar concepts in an external ontology to enlarge the results
— extracting old campaigns attached to these concepts and points of interest
— retrieving past interactions with these documents
— retrieving the emails of users of these interactions
— cleaning this list of emails, by checking late update requests (blacklist requests, etc.)

The appearance of one characteristics or another in a data source is unpredictable and may vary from one scenario to another. This unpredictability of variation in scenario clearly illustrates the need for a meta-model in order to fix the limits of data model definition. This meta-model will set the design guideline, and enable the adaptivity of the approach.

Needs analysis

The unstructured aspect of this scenario, and the multiple clients in the use case, have naturally led us to think of the solution in a distributed way. In order to fulfill

these objectives, we have defined a resource-oriented approach.

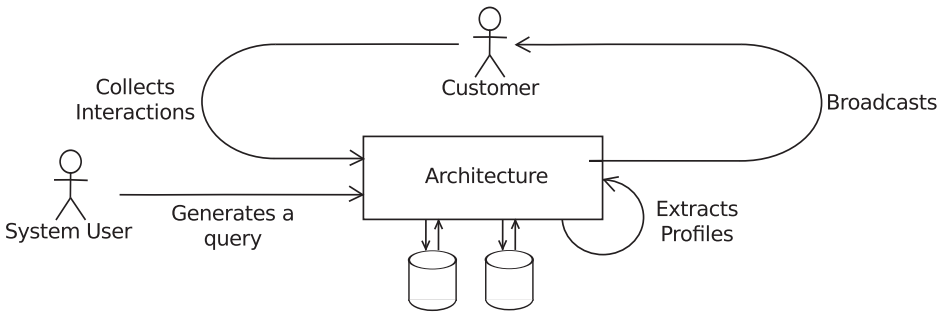


Figure 1.1 – Querying the system

Fig. 1.1 illustrates how our solution operates in the context of the Audience Labs scenario. It shows the interaction patterns between the system user, customers and the architecture.

The architecture provides a Web interface through which it is possible to submit queries. The system user creates a query through the GUI and extracts relevant profiles from the diverse data sources that are connected to the platform. The system user has access to several criteria including: age, gender, points of interest and related activities, socio-professional category, localization (country, region), browsers used, and so on. Once he has formulated the query, the architecture goes through the semantic annotation, and cleaning and integration tasks, as described above, and replies to the user query.

From campaign broadcast feedback, we transparently collect customers' interaction traces while simultaneously saving all contextual information about these interaction traces such as geolocalization, browser version, device type, and so on. By analyzing the customers' interaction traces, each customer will be attributed a specific profile that describes his areas of interests. The interest profile is updated on each of the customer's interactions. External data sources are also used to enhance the classification and evaluation of the user's points of interest. The semantic concepts that have been attached to data are reused to compute similarity between customers. The harvested data helps to retrieve all users that are more likely to be interested in a document, when the user performs a new query on the interface, thus closing the usage loop.

1.2 Scientific challenges

Usually, in classical integration systems, user input is heavy. Everything has to be hard coded, from data source interface to schema mediation. One can choose to rely on service interface to connect each data source, but there is still a huge need in terms of configurations. Another solution would be to extract data from data sources and store it in a common format, to construct views over this data to perform queries. However, each time a data source changes or is updated, stored data becomes obsolete. In these systems, every new use case needs a new approach.

The objective is to design a system which would be able, starting from a given number of data sources and a user-defined query, to identify which data sources could provide the desired concepts, and to interconnect these data sources in order to answer the query. In this context, there is a need for an approach which could easily adapt to data sources without having to recode the entire solution for each new scenario. First of all, it is necessary to identify a pattern in data source combination, and to identify the necessary steps of this global integration process. Secondly, having identified these tasks, build an adaptive framework to automate data integration from heterogeneous sources.

The global objective of this work is to propose a solution which is able to extract structured and useful information (i.e. smart data) from a set of predefined data sources. The solution we propose must be able to adapt to the different specificities of scenarios or data sources. The data samples extracted from the data sources have to be integrated in response to a set of user-defined conditions (pre-conditions, exclusion of some values, etc.) forming a data query. The generated homogeneous linked data set must respect a given quality level: it must not contain duplicates, malformed data, etc. In addition, our proposed solution needs scalability and responsiveness, i.e. we must respect a reasonable response time and must adapt to the scenario even when handling sensitive data sources. Sensitive data sources could be large volumes, data sources with a high latency, or data sources with a high update frequency.

In order to adapt to data source diversities, we have identified the following challenges and scientific locks to address :

1. How to guarantee a dynamic and transparent data source management. It must be possible to transparently add or remove a data source at runtime without any need of hard coded information. There is a need for a formalism to represent this diversity of data sources.

2. How to provide dynamic data processing. The solution needs to adapt at runtime to data sources that require different processings (large data volume, frequent update, latency).

Once we will have solved these data-source challenges, by introducing our first solution to automatically handle data access, we will have to implement the strategies we proposed in a software architecture, focusing on scalability and responsiveness. To do that, the solution must be scalable and support a large number of data sources, and a variety of data source types, while offering low response time. This will lead to the following new challenges:

3. Given a sequence of tasks to execute, how to adapt the orchestration of these different tasks in order to optimize the global process in terms of response time and relevance.
4. How to implement such a mechanism in a software architecture, providing the flexibility and the adaptability that our solution requires in order to adapt it to every kind of scenario.

Finally, when data access and combination challenges will have been solved, we will focus on quality of data, by considering the uncertainty that can appear in Web context, and try to solve the following questions:

5. What is the most relevant way to represent uncertainty on the Web ?
6. In what way could this affect Web browsing and automatic information retrieval (hypermedia navigation) ?

In the next section, we summarize the contributions we propose in this thesis to answer these challenges.

1.3 Contributions

After having analyzed the necessary steps to complete a global data source combination process, we propose the following contributions in response to these challenges:

1. First of all, we focus on data sources, and how data source characteristics influence data extraction, future usage and combination capabilities. We propose a meta-model that allows us to describe data source characteristics in a flexible way. In order to demonstrate the necessity for this meta-model, and in order to provide a basis for our further work on data access, we instantiate a set of

models based on our scenario, and demonstrate how these models can be used to adapt data processing according to data characteristics.

2. Then, based on these models, we define some specific data access strategies that will rely on data source characteristics to optimize data source access and data extraction. These strategies will provide the required dynamic data processing.
3. Secondly, we focus on these data access strategies based on data source characteristics, and propose a framework in order to create adaptive workflows that will help to complete the combination process. These adaptive workflow techniques will help to improve the required scalability and responsiveness.
4. The adaptability required by our scenario leads us to propose a distributed approach. So, after having analyzed the main different existing design patterns to build a distributed Web software architecture, we will define the outline of what we call a smart data architecture, where each component handles a specific part of the process. We envision the different components of the architecture, and rely on previously introduced data access strategies to propose architecture optimizations, and to improve the adaptability of our approach.
5. Finally, we focus on the concept of data uncertainty and how it can affect data combination on the Web. In order to improve the data quality of our approach, we propose a theoretical definition of the notion of uncertain Web resource and propose an interpretation model to access their uncertain representations. Then, we propose an algebra to evaluate this uncertainty in the context of classical hypertext navigation that allows us to combine data from several uncertain Web resources.
6. We rely on these models to develop a set of specific operators and algorithms to evaluate uncertain data queries.

1.4 Thesis Organization

The rest of the dissertation is organized as follows:

In Chapter 2, we provide the background knowledge which underlies our different proposals. First, we present the key concepts around the architectural style of the Web, from low level protocols to high level service technologies. We quickly present works related to semantics, presenting linked data and semantic enhancement approaches. Finally, we review the basis of data integration.

In Chapter 3, we focus on data sources and propose a declarative approach to represent them. We propose a meta-model to create data models and data source models. Based on these meta-model and models, we propose different data access strategies to adapt data source characteristics.

In Chapter 4, we rely on the previously introduced models and strategies to propose an adaptive resource-oriented architecture. We improve this architecture and show its adaptability by proposing an optimization according to data volumes.

In Chapter 5, we go further into our data quality objective by proposing an approach to define uncertain Web resources. We enrich it with an interpretation model and algorithms to handle uncertain resource composition, especially in the context of query answering by hypermedia navigation.

In Chapter 6, we provide a general conclusion and discuss some possible directions for future research.

Chapter 2

Background Knowledge

In this chapter, we overview the key concepts in the areas where our contributions take place. First of all, we present the architectural diversity of Web applications, from principles of HTTP and URIs to high-level service architectures. In a second time, we perform a quick review of semantic Web technologies and linked data principles, which are useful to improve automated (or semi-automated) data integration. Finally, we overview the basis of data integration, and formally propose a set of tasks which are required to perform a data source combination process. Each section presents a background of the aforementioned fields, detailing the required notions to introduce our approach.

2.1 Architectures of Web Applications

Nowadays, information systems are more and more distributed across the Web, physically or logically. We are now able to take full advantages of advanced Web technologies to interconnect our systems, formerly trapped into static core-oriented applications. Relying on this interconnection, data comes and goes between servers, hosted on the Web and available for any eventual stakeholders.

In this section, we review the evolution of Web applications, by presenting different paradigms and architectures. We present their principles, and some of the issues that may raise.

2.1.1 Architectures and Paradigms

The evolution and the opening of our information systems, associated with the growing diversity of devices, has forced us to change architectures and technologies to improve their interoperability. In this context, distributed architectures have become essential, since they propose highly decoupled features. In this chapter, we overview the key concept of Web architectures, by reviewing the technologies that are used at the different levels.

Service-oriented paradigms and architectures

Service-oriented computing is a computer software design paradigm, where an information system relies on separation of concerns to isolate operations. Web services are software components that are separated according to their logic, and made available on the Web by their provider. These services can be deployed on several locations, provided by different organisms, and relying on different logic. These distributed capabilities are combined to accomplish a final goal. This way, clients can fully concentrate on which results they want to achieve, without having to focus on smaller issues. In order to help create these distributed applications, Erl et Al. [Erl, 2005] defined several principles that form a methodology for engineering software components that are decoupled, cohesive and reusable, regardless of their location. In the following, we present these principles:

— *Loose Coupling* implies that a service does not need to have knowledge about the context or other services to work
— *Abstraction* makes components hide their implementation behind a uniform interface
— *Reusability* maximizes the effort of separating concerns into components in order to reuse them
— *Autonomy* ensures that components have control over their implementation and are independent from their execution environments
— *Composability* ensures that components can be combined in order to solve different kind of problems

These principles guarantee the independence of services, providing a simple way to quickly build applications by simplifying feature design. On top of that it allows parts of a final architecture to be developed and hosted by third parties. It helps to build service oriented architectures in a distributed and flexible way.

These service oriented architectures are based on three important parts: providers, consumers and mediators. Provider gives (or rent) an access to a part of its softwares, which will be used by a consumer (the client) that has a need for this specific feature. Interactions between clients and services are handled by Mediators.

Resource-oriented paradigm

Resource-oriented architectures have gain benefits respectively from the SOA principles and from the REST constraints [Fielding and Taylor, 2000]. This kind of architecture relies on the concept of resource, and respects the REST architectural style. According to REST principles, resources are the key abstraction of information. Everything that can be named can be a resource: a resource can provide a single object: such as a document or an image, a temporal service (weather in a city at a given time, etc.), a collection of other resources, or the connected interface of a physical object (connected objects, sensors, etc.).

According to REST principles, resource-oriented architectures is a software architecture where services exchange resource representation amongst HTTP transport. Services and resources are identified with self descriptive URIs. Doing so, resource-oriented architecture becomes the most generic and adaptable kind of architecture, with a complete respect of Web principles, making them well thought and relevant.

REST architectural style defines its principles as a set of constraints that architectures should respect in order to remain REST-compliant, i.e. to respect Web architectural style:

— Servers and clients are independent, user interface is client-side, storage is server-side (e.g. databases), doing so, we improve the *scalability* and the *portability* of clients across multiple platforms
— Communication must be stateless, such that each request from client to server contains all the necessary information and cannot take advantages of any stored context. This improves *visibility* (because request data transports request context, and also the nature of the request), *reliability* (reproductability of failures), and *scalability* (no context storage means that server will quickly free resources).
— Cache constraints defines that request responses should explicitly indicate if they are cacheable or non-cacheable, and for how many times. This improves *scalability* and *efficiency*, by reducing the amount of data exchanged through the network (induce by the statelessness of the application)

— These principles require a uniform interface in order to guarantee an optimal access to resources, this uniform interface is defined by the following principles, which we review later: resource identification, manipulation of resource through representations, self-descriptive messages and hypermedia as the engine of application state

— Finally, one last constraint is proposed in order to allow client functionality to be extended. REST allows download and execution of code on demand, under the form of scripts or applets, in order to improve system extensibility and to simplify client code by reducing the number of features required. Since this constraint can reduce the visibility introduced before (especially about the reproductability of errors), it is an optional constraint. This final notion allowed a whole area of programming languages to expand and evolve, through proposals such as EcmaScript and later JavaScript.

After having presented RESTful architecture constraits, we review the four principles which provide a uniform interface, as they are the key concepts we rely on to describe our data sources and build our architecture.

1. The main concept of uniform interface is the *identification* of resources on the Web. Each resource must be identified in a unique way, even if the data it represents can change over time. It is important to be able to identify every resource, even after a long time. This will be helped by the definition of URIs, which identify resources precisely in a network. Thanks to URIs, we access every resource in a unique way, and there must be only one way to access it. Different URIs must lead to different resources, even if they have the same representations. In our context, we will apply the same principles to our data sources. We must note that data sources on the Web are also identified by URIs. As an example, a tabular data source (like CSV for comma separated values) could be identified with a pretty simple URI such as: `http://example.com/path/file.csv`. In the same way, a database endpoint can be defined by a URI of the following type: `oracledb://User=user;Password=pass@domain.com?PollingId=polling`, containing more information about the source: such as the protocol, the authentication (not all authentication protocol), and other specific information.

2. Resources are managed by manipulating their representations. A resource representation is the information (and metadata) which characterizes the state of a resource at a specific time. In general, even if it is less precise, this resource

representation is called file or document. These representations are exchanged through HTTP, and according to the type of demand (control message), the retrieved representation could represent the actual (or a desired) state of the resource, at the time we request it. In the same way, to modify a resource, one new representation has to be created and sent (posted) to its location.

3. Manipulated representations are self descriptive, they contain the sufficient metadata allowing their understanding (e.g. format). As an example, the encoding is clearly indicated inside the document, so that clients can adapt the process. In our context, we want to take advantages from these resource capabilities. We assume that the information carried by data source should be sufficient to successfully process a data sources.

4. Finally, hypermedia as the engine of application state, implies that a resource should indicate how to be processed, and the available options in order to alter its state (deletion, edition, etc). Doing so, a client may instantly know which actions are available with this resource.

These two architectural paradigm, resource-oriented and service-oriented, are derived from the concept of services. In the next section, we present some general concepts about services.

2.1.2 Services

The W3C [2] defines a Web service as *a software system designed to support interoperable machine-to-machine interaction over a network.* Basically, Web services define a standardized way of integrating Web-based applications or features using different types of technologies, languages and formats.

WSDL/SOAP Services

The most known Web service approach, is called WSDL/SOAP services. This approach is an extension of the object-oriented paradigm in order to enable Web access to its objects and features, requested on demand. Web services are deployed software components, hosted on application servers, whose operations are accessible to the outside.

2. See https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice

These services rely on the Web Service Description Language (WSDL) for self-description and on the SOAP protocol for data exchange. WSDL description encapsulates different pieces of data about the service endpoints, such as available operations, input and output, with their data types. Service description is independent from service implementation and platform language. In order to exchange with these services (callbacks and responses), we rely on the SOAP protocol to wrap XML messages exchanged between service consumers and providers.

This type of service has received a huge support from the community, since they open oriented object software to the Web. A lot of approaches have been proposed to automate their conception from standalone applications, to allow their callback through electronic mail SMTP protocol, etc.

Despite these advances, this kind of services merely use HTTP as a transport protocol and do not correctly take advantages from its application-level features (provided by HTTP operations). Moreover, the WSDL language together with the SOAP protocol have proven to be very complex, which motivated the move towards a more simple conception of services, in order to respect Web principles.

RESTful Services

One of the reasons WSDL description is so important in the SOAP approach, is because each service has a specific interface, impossible to use without its description. RESTful services respect REST constraints, so they manipulate resources, and can be accessed through HTTP methods, making them really simpler to use.

In the Open System Interconnection model (OSI), which defines communication standards for all information systems, the lower level where data appears is the level 5 (session level). The session level is where communication is managed between hosts, and where data is transfered. Hypertext transfer (HTTP) is one of the main protocol used at this level, and it handles communications on the Web. Data is hosted on servers, deployed as resources and made available to the Web. Clients work with these resources, using HTTP requests to create, modify, delete and access them. A resource can be accessed with different operators, which are called HTTP methods, allowing to read, update, create or delete resources on a domain. In order to manipulate resources, HTTP methods work with representations which are the physical instances of these resources. According to *RFC 2616*[3], HTTP defines several types of request:

3. http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

— *GET* is used to retrieve the representation of a resource (at a given URI), it should not alter the resource, and without any information, it should retrieve the current state of the resource
— *PUT* sends a representation to a given URI (creates the resource or updates it if it exists)
— *DELETE* deletes a resource at a given URI
— *OPTION* retrieves the communication options available
— *POST* has a server-dependent behavior. It can be used to annotate a resource (by sending a partial representation), to send data to a URI for a specific process (classical form usage on the Web). In some cases, POST can also create a new resource, and in this particular case, the response message includes the URI of the newly created resource.

Relying on the previously introduced REST principles [Fielding and Taylor, 2000], a more conform approach has been proposed under the form of RESTful services. These services can easily be maintained and integrated with existing Web infrastructures since these services are essentially based upon Web standards, such as HTTP and URIs. The simplicity of this approach has made it widely adopted, especially for what it brings to Application Programmatic Interfaces (APIs) and how it simplifies combination with Web applications. Nowadays, a lot of Web 2.0 applications and almost all social platforms (such as *Facebook*, *Twitter*, *Instagram*, *Reddit*...) provide a public API implemented as a set of RESTful services.

2.1.3 Service composition

The powerfulness and the adaptivity of Web architectures, and how they have grown so fast, came from their ability to organize task management. Several approaches have been proposed to improve composition and orchestration of tasks into what we call workflows. A workflow is a tool that is used to modelize and automate task sequencing inside a software architecture. It is a representation of data flows (in terms of inputs and outputs) between the different components (in our context, services) of a system, in order to complete a business process. A lot of approaches have been proposed in order to model workflows, especially for SOAP/WSDL services.

Classical business process management

In order to combine services in classical architectures, several languages have been proposed, such as BPMN (Business Process Model and Notation), which is a model to represent business procedures as diagrams, modeling interactions between services, and BPEL (Business Process Execution Language) which is an orchestration language based on XML allowing to build service workflows executable by an orchestrator.

Whereas these standards fully integrate with SOAP/WSDL services, and are particularly efficient to represent a static workflow, they still presenta lack of adaptability. One major problem is the necessity to hard code a large part of the useful information. Thus, when a change of context occurs, workflow instantly becomes obsolete in cascade, and there is a need for a new workflow definition. Although many approaches have been developed in order to solve these issues, with various degrees of success, there is still a lack of flexibility.

RESTful orchestration

Since RESTful services had been proposed to simplify Web service development, it is necessary to define a workflow language able to interact with this kind of services, with respect to REST principles. Here we present some approaches that propose a solution to handle the composition of RESTful services.

Eckert et Al. propose a REST open workflow solution [Eckert et al., 2014], which relies on using open workflows [Stankovski et al., 2010] to represent the combination and orchestration of the necessary resources. Their approach proposes a workflow model for RESTful services described with Linked Data (RDF). They define three aspects which represent the basis of a workflow: Specification, Composition and Execution.

— The specification defines how services and resources are represented with a name, as well as several inputs and outputs.
— The composition will define how to connect these components, inputs to outputs.
— They describe workflow execution as the following: a service is started by sending (through POST) the configuration of an execution, i.e. a representation of the resource context; when a service or a workflow is invoked, a job instance is created, and this instance will send asynchronous information about process status. Once finished, the job URI points to the result of the execution.

They propose a workflow ontology based on the Open Workflow.

Rosenberg et Al. propose Bite [Rosenberg et al., 2008] a lightweight composition language to work with RESTful services and to create mashups. The Bite process model proposes a simple flat graph (except loops), with activities and links between them. The execution logic relies on conditional transition between activities. Bite workflows are process definitions, in which it is possible to describe the data and the control flow. Activities, such as basic HTTP requests, utilities (wait, execution of code, etc.), and helpers are sequentially executed, according to workflow definition. This kind of approach is well adapted for creating fast built mashups or Web compositions, but has a too static conception. Once created, workflows cannot be adapted, and any change of context will lead the user to produce another workflow.

Brzeziński et al. [Brzeziński et al., 2011] propose a Restful Oriented Workflow Language (ROSWELL) and the according execution engine. ROSWELL is a declarative business process language which support the use of RESTful Web services. This language can be used to described exchanges between REST resources such as resource-oriented workflows. Their approach relies on a logical syntax, similar to the *prolog* language, in order to represent interactions between resources. Authors strongly rely on RESTful principles to define their language, *uniform interface* is guaranteed by keywords to describe both description and invocation of resources, *connectedness* and *addressability* can be maintained with the possibility to split and recreate URL queries inside workflow description. However, the preservation of *statelessness* is not quite clear in the approach. The weakness of the approach stands in the fact that the workflow needs to be translated into another language to be functional, which limits the adaptability of the approach.

As a conclusion, the problem of restful resource composition in classical workflows is the lack of respect of restful principles in standard approaches. This problem is also due to functional point of view of classical services, i.e. we invoke a service to execute a process, which restful resources are not. We need to extract the functional part of services and there is a need for an representation of this functional behavior inside the workflow.

What comes out from this analysis, is that we need to configure our resources, to generate the configuration of execution context in order to prepare our execution. Workflow languages have to implement data handling inside the workflow, i.e. it is necessary to process the data, decompose the resources and create the new configuration instances directly in the workflow. Then the workflow will be able to fetch the

resources through HTTP requests. Most of the existing languages only propose an SOA overlay over REST resources. As an example, in these approach, we do not see the use of HTTP status code to improve the process (1xx codes for standby, 2xx codes for OK, 4xx codes for wrong configuration, 5xx to perform fault tolerance).

2.2 Semantic Web and Linked Data

In this section, we perform a review of semantic Web and linked data principles. We review some approaches that enhance semantics around service description and composition, and finally, present existing approaches for semantic enhancement of data sources.

2.2.1 Linked Data and Semantics

One of the most significant descriptions for the semantic Web objective is from [Berners-Lee et al., 2001] and says: *"the semantic web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users."*. This is the first objective of semantic Web: to bring more meaning to data in order to improve reuse capability of this data. The Web of data [Bizer et al., 2007] (or linked data) is a concept derived from the semantic Web [Berners-Lee et al., 2001] in order to generate a machine-readable Web as an extension of the human-readable Web. The main objective is to generate a Web exposing and interlinking data previously enclosed within silos. Linked data initiative has led to the publication of billions of pieces of data, transforming the traditional Web of documents into a Web of linked data [Heath and Bizer, 2011]. In the Web of linked data ideal, data consumers, i.e., users, developers and their applications, make use of links between pieces of data to discover related data stored in remote servers, augmenting their added-value and enriching user experience. The principles of the linked data initiative are the following [Bizer et al., 2009]:

1. Use URIs as names for things. This will extend the scope of the Web from online resources to encompass any object or concept in the world (especially helpful for Web of things)
2. Use HTTP to look up at these URIs. This principle enables the dereferencing of these URIs over the HTTP protocol into a description of the identified object

or concept.

3. When someone looks up a URI, provide useful semantics (representation of data meaning), using standardized formats, traditionally RDF. The purpose of advocating a standardized format is to ease the interoperability and contribute to the scalability, as with HTML in the Web of documents.

4. Include links to other URIs, so that users can discover more data. This principle follows the idea of hyperlinking documents in the traditional Web. The main difference is that in linked data, links are typed, e.g, two persons can be linked with a hyperlink of type `friend` or `relative`.

All the metadata representing functional and non-fonctional information about the data, i.e. the meaning of data, is called *semantics*. In order to represent semantics, several notions are used, most important being the notions of graph and concept. A graph is a set of resources (which we call concepts) that share relations. As an example, we could define the two following concepts: `animal` and `dog`, which are connected each other by the following relations: `dog` is a type of `animal`. In the RDF format, which is used to describe conceptual graphs, this statement is called a triple, and is always defined between 3 elements: subject, predicate and object. Here, `dog` is the subject, `animal` the object, and `is a type of` is the predicate. An object from a relation can become the subject of other relations, as an example: `animal` is a `living thing`. In the same way, subjects can become objects: `Sammy` is a `dog`. Finally, predicates can be any type of property between two resources: `Sammy` lives with `John`, `John` wears a `coat`.

Since concepts are resources, they are identified by URIs, manipulable through HTTP, etc. In order to organize knowledge about a specific domain, ontologies have been proposed. Ontologies regroup the possible concepts from a domain, and the different types of relation they can share. As an example, the FOAF (friend of a friend) ontology [Brickley and Miller, 2007] was originally built in order to describe people relation on the Web, while the QUDT [4] ontology was developed in order to build a consistent and standardized vocabulary about Quantities, Units, Dimensions, an data Types. This ontology can be used to automate the conversion of dimension from a unit to another.

Since these principles were proposed we have witnessed an outstanding growth in terms of ontologies and datasets, see for example DBPedia, a machine readable version

4. See http://www.qudt.org/

of Wikipedia knowledge (http://dbpedia.org/); Geonames for geographical informations about countries, etc. (http://www.geonames.org), MusicBrainz for knowledge about music, artists and records (http://musicbrainz.org/) or DBLP for scientific publications (http://www.dblp.org/). In addition, the appearance of new standardized format, like N3 [Berners-Lee, 2005] or JSON-LD [Lanthaler and Gutl, 2012], which allows to annotate JSON with linked data concepts. JSON-LD has become a Web standard as recognized by W3C [5].

2.2.2 Semantic annotation

In our data integration context, semantics could provide useful information about the manipulated concepts. Relying on these concepts could help to make a matching between data sources, which is specially useful when trying to combine sources. Therefore, there is a huge effort from communities to propose approaches that help concept recognition, and semantic annotation of data sources. In the following, we overview some of the approaches that have been proposed in order to automate semantic annotation of documents, or semantic concept recognition.

Furth et Al. [Furth and Baumeister, 2013] propose an approach for the semantification (enrichment with semantic description) of technical documents. It relies on different working steps over the document to be enriched. First, it converts the document into a standard format, then it splits the document into segments, and applies natural language processing techniques to the document parts in order to extract a set of ranked concepts. This set of concepts represents the main subject of the document. The strength of this approach is that it does not require a huge set of training data to provide a classification. They provide a performance evaluation tool by adding a manual step allowing domain experts to review results.

Venetis et Al. [Venetis et al., 2011] describe a system that recovers semantics from tables existing on the Web. Their solution relies on the help of a set of millions of other tables to identify the role (or subject) of each column/attribute. The solution stands on performing similarity computation with the corpus of tables and extracting entities with the help of natural language processing over table values. The main drawback of this approach is that it requires a huge amount of objects in the corpus to analyze. Moreover they rely on millions of English-speaking documents to build their relation and entity extractor, which severely decreases the scalability of their

5. See https://www.w3.org/TR/json-ld/

approach.

In **TARTAR** [Pivk, 2005], Pivk proposes a solution for extracting knowledge from tables picked up from the web. The solution, based on Hurst's table model [Hurst, 2001], relies on the analysis of table across different points, including physical, structural, functional and semantical dimensions. The first step is a regular matrix extraction from physical dimensions. Analyzing the structure allows to determine table reading orientation, to dismember a table into logical subpart and to resolve types. The functional table model helps to deduce the role of each cell. A last step provides semantic concepts and labels for each column, using external tools such as WordNet lexical database. These models and concepts allow to populate a domain ontology from table rows.

The previously introduced approaches show the effort made towards automatic semantic annotation of data. For the sake of simplicity and effectiveness of our approach, we decided to rely on a semi-automated technology in order to add semantics to our data pieces. We present in the following different solutions that rely on the design of mapping files in order to directly generate Linked Data from data sources. There are different approaches in this area, technical approaches as well as theoretical approaches.

Han et Al. present in **RDF123** [Han et al., 2008] an open-source tool for translating spreadsheet data to RDF. They rely on a column-based mapping, where a set of expressions represents the structure and orchestration of cells in a tabular row. They define a whole language to describe these expressions, allowing to define control branches and data manipulations. The generated mapping file containing the set of expressions can be serialized as RDF and placed as a link in spreadsheet metadata, for reusability.

The **Triplify** [Auer et al., 2009] solution proposed by Auer et Al. allows to attach to a pre-existing system a module that will publish data as a Linked Data store. The solution creates a set of configuration files and associates semantic concepts (URIs) with SQL requests. Once the configuration has been created and the module has been integrated to the system, the module is accessible as a web page within the application and will be registered with a central repositories of data sets. This solution does not allow any flexibility, since each configuration is hard coded in the system. Otherwise, the system does not provide any computational access, and access is only accessible as a generated HTML/JS interface. Accessing the generated linked data pieces in order to manipulate them is not possible without changing the core of the product.

Bizer et Al. propose the **D2R** [Bizer, 2004] platform, which gives access to relational databases through a SPARQL endpoint. The platform relies on a virtual RDF graph, which associates concepts and relations to SQL requests. D2R gives access to databases relying on a N3 mapping files which can be generated by one of the provided platform tools. This tool relies on inherent database structure (foreign keys and relations), to deduce relations between table fields. In order to make a database available, the configuration has to be generated and the platform has to be launched through an application server.

We rely on these latter approaches to perform our semantic annotation task, by automatically creating the mapping expression with help from external and third party services for semantics extraction and concept recognition.

2.3 Data Integration

In this section, we review the basis of data integration focusing on data source combination. In this context, we only consider the integration task that consists in combining data from different sources to provide a unified view to the user [Lenzerini, 2002]. In our approach, we focus on a graph-based composition of documents, where we combine data from data source sequentially, and rely on common values to combine data sets, in the same way that a join would work on pivot fields. In **Levy et. Al** [Levy et al., 1996], authors argue that knowing data source description (viewed as a set of inputs and outputs), should be sufficient to combine the data it contains. We rely on this notion to propose a step-by-step approach to data source combination. Finally, we propose a review of state-of-the-art architecture for data integration.

2.3.1 Data Integration Processes

In this subsection, we describe the complete integration process as we see it, starting from a data source. We separate each steps of this process, in order to isolate each concerns.

The starting point of our integration process is a set of data sources from which we want to combine the data. As an example, we want to combine a data source containing user profiles (email, name, address, etc.) hosted on a public server, with other data sources containing another list of user profiles which associates different other information with email addresses (such as phone number, purchase history, etc.).

We propose a formal representation of these tasks, in order to help our proposal design.

Data Extraction

The first task is the extraction from data source (download, callback, request). This task objective is to retrieve all or part of the data contained into a data source.

Each data source has several characteristics (URI, format, etc.) which should be necessary and sufficient to extract the data it contains. In our example, one data source is a *CSV* file hosted at a *URI*. In a data integration approach, the data extraction process is one of the most important step, during which every data source format requires a specific process. In the next chapter, we focus on this task, by proposing models to handle data source diversity. In this chapter, we will call this task *Data Extraction*, and we do not focus on all the challenges it raises, since we address this challenges later.

We consider a data source DS_a, defined by a set of characteristics. We define a download function $Download()$ that extracts a quantity of data D from a data source DS_a.

$$Download(URI_a, S_a[, K_a[, Q]]) = D_a$$

where URI_a, S_a represent the data source DS_a (URI and Model) and D_a the data extracted.

An access function can accept optional parameters (query for databases, authentication parameters etc.). In this case, the download function handles the specific authentication or secure protocol to access the data source where K_a represent the authentication information required (user/password, ssh key, api key, etc).

In case of a sub extraction, when only a part of the data is retrieved (database query), the extraction process requires additional information about the data to extract. Q contains the information required to extract only a sub part of the data (offset and limit for an API, query for databases, etc.)

Data Format

Considering that data has been retrieved, there is a need for a common format, in order to compare what is comparable. We assume that each data set can be trans-

formed, with more or less difficulty into a key-value structure. Given this assumption, we choose to use a key-value model as pivot format to manipulate data.

In order to be processed, data needs to be transformed from its raw extracted format to a format we can manipulate. We define a decoding function $Decode()$ which will transform the data into our standard format.

$$Decode(D_{a,r}) = D_{a,f}$$

where $D_{a,r}$ is the data extracted into its raw format and $D_{a,f}$ is the transformed data.

Semantics

In order to combine data sources, semantics is important, because, it informs about the meaning of fields, what fields represent. Each data source represent a semantic graph, where data fields and keys are linked together according to functional properties. This graph is used to express the meaning of each piece of data represented in a data set. The semantics underlying databases, as an example, can be observed looking at the relations between tables.

According to Linked Data principles, each semantical information can fit into a triplet: (subject, predicates, object). A user which has an email address can represented as: (user, vcard:hasAddress, "user@mail.com"). In our approach, we define G_a being the semantic graph of a data source DS_a.

```
PREFIX al: <http://restful.alabs.io/concepts/0.1/>
PREFIX xsd: <http://www.w3.org/TR/xmlschema-2/>
SELECT ?email_value ?campaign WHERE {
    ?email a al:email ;
           al:has_email_value  ?email_value ;
    ?clic  a al:clic  ;
           al:clic_email  ?email ;
}
```

Listing 2.1 – Example query from the scenario

As an example, Listing 2.1 present a SPARQL query from our scenario. In this query, several data source subgraphs are covered. The set of triples:{ ?email a al:email, ?clic a al:clic, ?clic al:clic_email ?email} indicates a relation between two concepts, the first being a $al : email$, the other being a $al : clic$. This subgraph is covered in the customer activities database, which attach user activities (here, clics) with customers (identified by emails).

According to use cases, and depending on data sources, the amount of necessary semantics to be able to combine source can vary. In some cases, data only need to be annotated with semantics. In other cases, it requires a full data transformation into linked data.

In order to transform the data extracted from a data source into I_a, an instance of its semantic graph/subgraph, we define a mapping function $Transform_a()$ which is defined as:

$$Transform_a(D_a, G) = I_a$$

where D_a is the data extracted from data source DS_a, G is the linked data graph, and I_a another linked data graph produced from DS_a.

When semantic annotation is sufficient, we define $Annot_a$, which annotate a data set with linked data:

$$Annotate_a(D_a, G) = DA_a$$

where D_a is the data extracted from data source DS_a, G is the linked data graph, and DA_a the data set annotated with semantic concept from the graph.

Combination

Once data has been extracted and semantically enhanced, it can easily be combined into a new data set. Two (or more) semantically enhanced data set can be integrated. During this task, concepts are compared and aligned each other, in then the data sets are combined to form a new one We define an integration function called $Combine()$ which takes as input the data sets that have been previously annotated and combines them into a new one.

$$Combine(G, D_a, D_b, ...) => D_{mix}$$

where G is the semantic graph of manipulated data and (D_a, D_b, ...) are semantically transformed extracted data from data source (DS_a, DS_b, ...). Finally, D_{mix} is the smart data set result.

The function relies on graph instances to link the concepts of the different data sets with each other. It analyzes the data pieces which have to be combined and provides on-the-fly mediation by fulfilling the data piece conversions and transformations with help from a set of predefined mediation processes. Based on the domain ontology, the integration function combines the data sets based on their common concepts.

The function performs concept matching to link concepts and perform a Cartesian product over matching data pieces (i.e. similar to a database join with a pivot). Before performing the combination, the integration function analyzes data and detects heterogeneities, providing mediation based on our previous work with the DMaaS approach [Mrissa et al., 2013]. The DMaaS approach proposes an architecture that solves data inconsistencies in service compositions. The approach focuses on service descriptions to analyze conceptual compatibility, and resolves conflicting aspects with help from mediation services.

Data Consistency

One important part of our work, is to provide qualitative data. In this context of data quality, we identify different types of data problems:

— Duplicates, when a data set contains several copies of the same piece of data

— Contradictions, when a data set contains two pieces of information that can exists together, we call this data *mutually exclusive*

— Noise, when a data set contains information which it should not contains

To remove these malformed pieces of data, we define another function $Clean()$ that removes the malformed part, noise and inconsistencies that may appear in a data set D_a after or before processing.

$$Clean(D_a) => D_{a,clean}$$

We identify another specific case, when according to a scenario, data should be processed to remove data pieces responding to specific (user defined) conditions. This function takes as input a set of conditions, which are called $filters$.

$$Filter(D_a, filters) => D_{a,clean}$$

```
PREFIX al: <http://restful.alabs.io/concepts/0.1/>
PREFIX xsd: <http://www.w3.org/TR/xmlschema-2/>
SELECT ?email_value ?campaign WHERE {
?email      a                    al:email ;
   al:has_email_value  ?email_value ;
   al: blacklist_status  ?status .
?clic       a                    al: clic ;
   al: clic_email       ?email ;
   al: clic_date        ?date .
FILTER (?email_value != 1 &&
        ?date >= "1411477450"^^xsd:date)
}
```

Listing 2.2 – Another example query from the scenario

As an example, Listing 2.2 present a SPARQL query from our scenario, in which two filter are applied. In this query, we filter emails making sure they does not have a blacklisted status, and we also filter the date associated with our user activity.

The different processing tasks defined here will help to complete the tasks that participate in the integration process. In the following, we present how these functions can be combined into different processing workflows depending on the characteristics described in the data source and data models.

2.3.2 Architectures for Data Integration

The study and design of architectures to automatically integrate data from diverse resources and produce smart data is currently a hot research topic explored by the community. Smart data has caught the interest of the community as a natural development after the interest around big data. The objective with smart data is focused on producing high-quality data that is directly useful to the users, instead of big data approaches that focused on building solutions to process massive data quantities.

Dustdar et al. present a *peer data network* architecture in [Dustdar et al., 2012], where data sources are independent databases. They propose an infrastructure relying on data services where tasks are separated into levels, isolating data management and service integration. Their solution focuses on quality of data and provides service-based optimization, such as peer replication, to resolve data issues. However, the paper does not address data heterogeneity problems, assuming that schema mapping is sufficient.

In *QuerioCity*, [Lopez and Kotoulas, 2012] presents a smart platform to catalog, index and query heterogeneous information from complex systems such as city data portals [6]. They focus on data integration and semantic annotation problems, mainly on issues related to scalability, unpredictability of changing data and impossibility to fully automate the integration process. The proposed approach clearly distinguishes between the data integration and data consumption tasks. In order to harmonize data usage, data fields are converted to a standard format, annotated with metadata and aligned with public ontologies (Dublin Core [Weibel and Koch, 2000] or FOAF [Brickley and Miller, 2007]). The effort is placed in management of extracted

6. Such as Dublinked http://www.dublinked.ie/

data, data access challenges and the needs which arise are not part of the scope. The approach focus essentially on extraction of meaning and semantics from datasets as well as provenance in order to provide a harmonized dataset. They do not provide any information about data source classification, and assume that sources have to meet the request format that the architecture supports.

In the same way, several approaches has been proposed to provide decision support systems to perform Analytics-as-a-Service [Delen and Demirkan, 2013], such as Sun et Al. [Sun et al., 2014], which propose to rely on a service oriented architecture to provide data analytics over business data, relying on cloud and service technologies to perform the discovery of data analysis services, but their approach does not address the issues of how to access and characterize the data sets that will be processed. Many of these big data approaches propose methodology and techniques to improve data processing, and mapping algorithms [O'Donovan et al., 2015]. So far, none of these approach address the data source issues, which is an important topic, since the capability to extract and reuse data will come from these resources.

We rely on these approaches to build our proposal, improving the reusability and loose coupling through usage of linked data services, automating the linked data efforts and re-structuring the different layers of the platform according to our needs that require reasoning about data and proposing a loosely-coupled approach for the different tasks to perform on data sources.

2.4 Conclusion

In this chapter, we present the main concepts about Web architecture and Web services, review the basis of data integration, and introduce the necessary concepts about semantics. Later on, we rely on the strengths and weaknesses of all the presented approaches to build an original and relevant proposal, following our global objective, which is to provide an adaptive resource-oriented architecture in order to combine data from heterogeneous data sources, based on semantics.

We take advantages from resource based approaches to build a generic, adaptive and flexible approach. We rely on a generic architecture to perform the different tasks that are required to combine the content of the different data sources. After that, we rely on different existing techniques that allow to enhance data with semantics, to identify or describe the meaning of extracted data pieces, by transforming or annotating data.

As an example, the JSON-LD format allows us to make a gateway between data and linked data. In the following chapter, we introduce our first contribution, which focuses on the different data sources capabilities.

Chapter 3

Model-driven Data Source Management

3.1 Introduction

In this thesis, we focus on a solution to help combining data sources into homogeneous linked data, in response to a specific user-defined query. We propose a solution based on a Web software architecture, to handle this publicly available data. While this objective raises interest, automating the integration process remains a complex task, and there is still a lack of generalized approaches, particularly when focusing on the diversity of data sources. Indeed, one major difficulty of automating data source integration is to handle data source access, and differences in data sources require different approaches. Here are some of the many essential questions we ask about the data sources, when trying to access them, in order to retrieve the data it contains:

1. Where is the data source located ?
2. What is the procedure/operation to access it ?
3. Does it require an authentication ?
4. What is the language/format used to request the source ?
5. What is the format in use to represent the data itself ?
6. What does the data represent ?

3.1.1 Objective and scientific locks

Answering these questions will help to understand data sources, and how to adapt clients. We refer to these answers as *data source characteristics*. Data sources can have several characteristics (URI, format, authentication, etc.) that define the strategy in use to interact with them.

On top of that, there are several data source characteristics that will affect or alter data access: huge data, latent data source, or high frequency of update. This has to be taken care of to perform relevant data source management. As an example, extracting a large volume of data has a cost, and affects execution time. In the same way, accessing a data source with a high latency requires specific techniques, since every request to the source will increase the global execution time. In this case, it could be interesting to setup temporary storage mechanisms, such as cache.

In this chapter, we address the following scientific challenges as sub-problems of our global integration process:

— How to enable a dynamic and transparent data source management. There is a need for a model that will carry data source capabilities, in order to lower the quantity of hard coded information.

— How to provide a dynamic data processing: there is a need to provide an adaptive strategy at runtime according to data source characteristics. Different characteristics require different processing (large data volume, frequent update, latency)

— Finally, how to guarantee the scalability and responsiveness: in the end, our approach must support a large number of data sources while offering low response time. In order to do that, there is a need to identify the characteristics that could create latency and slow down the process.

3.1.2 Motivating scenario

In order to build our proposal, we rely on a sub-scenario from our our global company scenario. In this chapter, we only focus on the challenges related to data sources. We introduce the following data sources, each of them presenting several specific characteristics.

1. an internal linked service, i.e. consumes and produces linked data, providing access to a small data set that describes the business data of the company. Data pieces that come from this source are subject to privacy constraints.

2. a SQL endpoint to a database that contains millions of tuples (very large volume of information) with no semantic annotations, representing the customer base. This data source has a low update frequency.
3. another SQL endpoint to a database that records user daily activities (high volume of changing data) with 10.000/20.000 new tuples per day (updated regularly, so it requires freshness)
4. a RSS stream that contains user update requests, it mostly contains data which has to be saved or removed from data results (blacklist information)
5. a set of external sources, represented by a set of APIs (Twitter, Facebook) that help construct interest profiles, as well as a Dbpedia SPARQL endpoint for concept manipulation.

The scenario objective proposes to integrate the data from these data sources according to a user request. This request can involve one or several sources, contain specific data filters and/or required information. As an example, the following textual requests, which can represent simple demands or more complex ones:

– "Retrieve the first name, last name and email address of a customer for which we recorded an activity within the last seven days."

This query example involves 2 data sources: the customer database and the activity database.

– "Retrieve the last name and email address of all the customers, for which we have recorded an activity on any campaign whose subject is 'flower' or equivalent, within the last 4 days."

This query is far more complex, and involves 4 data sources: business data linked service (for retrieving campaigns and associated subjects), activities and customer database, and finally, an ontology to lookup at concepts that are similar to *flower* (plants, bloom, ...).

3.1.3 Contribution

In this chapter, we address different issues, by focusing on data sources aspects. The data sources shown in our scenario present different characteristics, which severely impact how data processing should be performed. Data samples and sources can be difficult to process due to these characteristics. As an example, the problem of manipulating large sources could be solved using cache techniques, but it requires to

take care of data freshness. Moreover, the appearance of one or another characteristic in a data source is highly unpredictable and may vary from one scenario to another.

This unpredictability of variation in scenarios clearly illustrates the need for a meta-model that will set the design guidelines in order to provide the necessary adaptivity to our approach. To build our approach, we design a generic metamodel and rely on this metamodel to construct several other models focused on data source description, access and processing. Each data source will be described with a combination of specific characteristics that allow to perform data access. Our meta-model allows to describe data sources in a flexible way and to generate models that in turn will provide adapted data processing depending on the scenario. The provided scenario demonstrates how the different models (data source model, data model and data access model) enable specific data processing to handle data source characteristics.

Finally, by focusing on these data source characteristics, we should be able to define different data access strategies, and de facto be able to adapt our process for each data source. This chapter is organized as follows: Section 2 presents a quick review of related work, by presenting approaches that enhance data source management and/or access. In section 3, we define our meta-model to help data source characteristics representation, and illustrate this model by instantiating a set of scenario based data models and data source models. In section 4, we present our data access strategies, and show how it could improve the general data integration process. Finally, we discuss our approach.

3.2 Related work: Data source access

There are plenty of approaches that propose data integration based on data sources, but most of the time, these state of the art approaches will focus on techniques to help data source manipulation. Classical approaches that solved data source diversity issues will require to wrap data sources with different types of overlay, to work with the same type of endpoint. We identify several types of solution, with approaches described as follows:

— using views to create ready-to-use data access over databases (such as Local-as-View and Global-as-View approaches) [Halevy et al., 2006],
— performing abstraction of heterogeneous data source, using structured database or XML to uniformly represent data [Pontieri et al., 2003, Rosaci et al., 2004]
— doing transformation into ontologies or RDF [Alani et al., 2007, Bizer, 2004] in

order to work with reasoners above this generated data

— exposing data sources as services [Cramer et al., 2004] or allowing data access through APIs [Groth et al., 2014] in order to work with these data sources in the Cloud, as an example.

These approaches require a huge amount of work upstream in order to adapt the solution to any kind of data sources. We believe that since data sources are available on the Web, we should be able to access them directly, using the right client, without having to transform the source.

The Atos Worldline company proposes a solution for the automatic management of data coming from any kind of source, called SmartData.io [AtosWorldline, 2013]. The application provides a RESTful API through which it is possible to publish data sources and data streams presented in several formats, such as CSV, PDF or RSS. Data is then extracted from files, converted into a pivot language (which is JSON) and then preprocessed by specific applications, which can be internally developed by the company or externally developed by third parties. Depending on the application, the extracted data is filtered by applying patterns or by combining it with additional data. Doing so, only the necessary and correct data is stored into the infrastructure. The presented architecture has very interesting aspects especially about automatic data processing but it requires the solution to host all the data sources, which are then deployed under APIs.

Apache Metamodel [Apache, 2013] proposes a data access framework, which offers a transparent rich query interface to different types of data stores, which does or does not usually provide this kind of request abilities. The framework provides a uniform connector to many types of data store, including several formats for databases, data files or objects. They propose a scripting language for processing updates and transactions via APIs. The architecture provides a uniform driver approach for requesting data sources, but acts as a static process where each source type has a particular adapter.

While these approaches do not address challenges related to data combination and data source configuration at runtime, they provide a universal access to different types of data sources and some technical solutions to request these endpoints.

In our context, we make the hypothesis that the best way to access and process data sources is to rely on their original capabilities. We decided to focus on data source characteristics in order to propose adaptive data access strategies. In order to propose this solution, we focus on the challenges related to data source diversity.

3.3 Data source models

As introduced before, data sources contain or produce data in their own format, responding or not to standards, e.g. CSV or XML for structured files; tuples for databases; JSON or XML for streams and services. However, according to scenarios, characteristics can appear and disappear, and these characteristics can, according to scenarios, be represented amongst different ranges and domains. There is a need for a meta-model that will allow us to describe the characteristics of data and data sources with models according to scenarios.

In this section, we describe the following models.

— our metamodel to describe data source model, setting data source model limits, and obligations

— a data source description model (instantiated from metamodel) to describe data source characteristics (physical characteristics and access needs)

— a data description model (which respects the metamodel and follows the data source description model) which is used to describe the data that will be extracted from the data source. The information from this model overrides the information of the data source description model if any. Indeed, specific data characteristics prevail on generic data source characteristics.

— a non-structured model, as a set of directives, called data access strategies. These are rules that will apply when accessing the data sources. We define these strategies as prerequisites to perform data extraction

3.3.1 A metamodel for describing data sources

Since data source processing capabilities depend on their characteristics, we build our adaptive integration approach on a flexible data source representation. To do so, and as presented before, we define a meta-model for describing data source models that could easily adapt to any use case.

Characteristics appear at different level. In this meta-model, we separate them in two groups: data source characteristics (i.e., *physical* characteristics), and data characteristics that will apply to the extracted data, instance or schema. Doing so, we separate our model in order to clearly identify the information which are required before data extraction, in order to help the process, and after the extraction in order to help data manipulation.

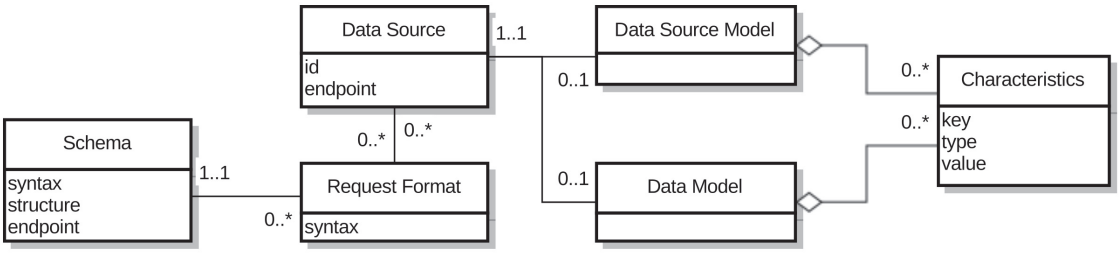


Figure 3.1 – Data source metamodel

Having performed this separation, we extracted the smallest subset of mandatory information. We identify the couple $< URI, requestformat >$ as the smallest necessary set of information required to access a data source, some of them will require additional information, but most of the time it will be sufficient to perform a simple access to the data source (extraction, management, etc.).

The *URI* is one of the most important information about a resource. By definition [7], this character string permanently identifies a resource inside a network. One URI identifies a unique resource, and it is the main entry point to access it (or the data it contains). Every data source should have a URI, and this URI can contain relevant information about the resource.

The *request format* characteristic is represented by a *syntax* attribute (e.g. XML, JSON, SQL) and a *schema* defined by three attributes: *endpoint*, *syntax* (e.g. XML, n3, JSON) and *structure* (e.g. RDFS, XSD, JSON Schema).

On top of that, the meta-model includes a set of core characteristics forming the data source model and the data model. The *Data source model* is defined as an object that contains all the necessary properties and attributes, that will be used by the client to request the data source. These properties are scenario-specific. The *Data model* defines the set of properties and meta-data information that apply directly to the extracted data. These information are specific to the extracted instances.

Figure 3.1 presents this meta-model. While we illustrate the use of this meta-model in the context of our scenario, the former remains applicable to any new data source characteristic and other scenarios.

7. RFC 3986: https://tools.ietf.org/html/rfc3986

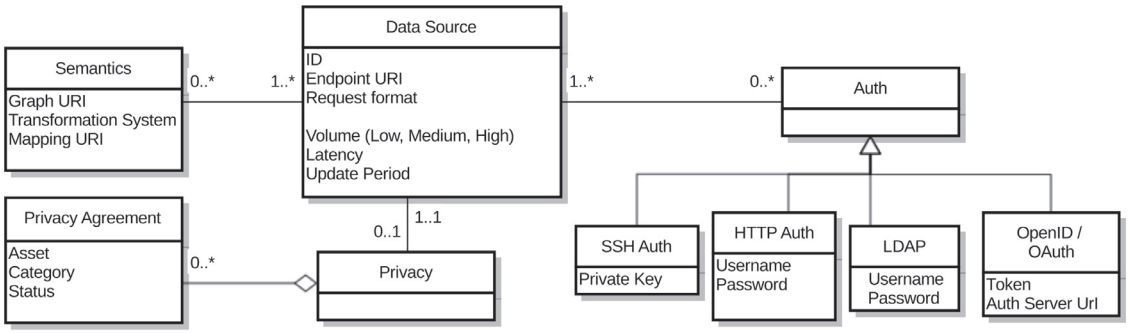


Figure 3.2 – A data source model based on our scenario

3.3.2 Data source description models

Relying on the meta-model presented above, we create an adapted data source model that presents the characteristics which are shared or specific to the different types of data source presented in our scenario (cf Section 3.1.2). We consider our data sources as defined by a set of the following characteristics: Data Source ID, Endpoint URI (data access transparency), Request format (SQL, SPARQL, JSON, XML etc.), Data volume, Latency, Update period, Authentication, Semantic and Privacy agreement [Truong et al., 2011]. Fig. 3.2 presents our scenario-based data source model.

The *URI* characteristic identifies the data source and contains the necessary information to enable the interaction with this data source. An URI is composed by at least: a *protocol*, a *domain* and a *resource*, e.g `file://localhost/home/file1.xml`. The protocol specifies the source communication procedure to apply, such as HTTP, FTP or SGBD connection. URI can also contain authentication information, port number and query parameters. URI can transparently identify any resource, a HTTP URI for a web resource, a file on local system, a database URI, etc.

Request Format defines how to interact with the data source. Most common request formats are SQL, SPARQL, JSON and XML.

Update frequency indicates the recommended average duration between each request to a data source. An update frequency of 0 means that each request may retrieve different data. The update frequency value has an impact on cache or synchronization possibilities.

Volume represents the global quantity of data that a data source manages. Depending on the volume of the data source, specific data access strategies can be adopted. According to the specific strategies to access data, we defined different volume inter-

vals. A `small` volume data source can be accessed directly (less than 20 Mo of data). A `medium` volume data source (from 20Mo to 200Mo) requires cache in order to handle eventual delays. A `high` volume data source (more than 200Mo) may required synchronization systems or big data mechanisms such as map/reduce.

Latency represents the average network time required to obtain a response message to a request on a data source. This value is maintained and updated regularly by statistical measure of delay.

Authentication describes the data source access restriction. This attribute can take different values, or can be disabled if data is publicly accessible. Common values are HTTP-auth, where access is granted by server directives over username/password verifications, OAuth or OpenId, where authentication is handled by a third party server, or SSH public/private keys. In some cases, auth parameters can be specified in the URI, e.g. `http://user:pass@test.com/`.

Semantics aggregates the information required to perform the semantic transformation from raw data to linked data. The semantic description contains: an URI of the linked data graph that describes the data model, an URI of the mapping file that gives information about required data transformation and an attribute identifying the system used to perform the transformation.

Privacy agreements define whether or not data is limited to a specific usage context, according to a set of conditions. Agreements can, as an example, avoid to provide a piece of data to a third party system, or prevent any modification or commercial use of a data piece.

3.3.3 Data description model

At the data level, there is a need for a model to describe data characteristics. Based on our scenario, we define here a data model that allows to characterize data sets and instances with specific attributes. The model we devised contains the following attributes that describe data instances: Privacy, Validity, Semantics and Filters. These attributes can be associated with either data tuples or globally with the whole data set.

A set of *privacy* attributes describes privacy requirements that has been given to data values by the data owner. As an example, a user who provides an email address, solely on the condition that she or he does not receive any email, requires a specific data agreement to be associated with the data value.

Validity specifies the lifetime of the data we extract from the data source, in other words it give the date after which the data will be considered as unusable or obsolete. Validity is different from update frequency, as a data source can specify an update frequency of an hour, and specify that the provided data is valid until the end of the year. We introduce a property named *Validity* that represents data lifetime, this information is related and stored with extracted data. This parameter indicates the amount of time after which the data will be considered as unusable or obsolete.

Semantics are conceptual metadata which are associated with a data set. Data semantics can be provided together with the data, when accessing the data source, or updated later with a semantic annotation process.

Filters are scenario-based specific attributes, which specify the values in the data set in terms of quality. Filters can specify a detected malformed piece of data, or a forbidden value.

The data constraints introduced in the data model always override the data source characteristics. As an example, a privacy agreement in the data model can specify the recipient allowed for a piece of data, but a data source level privacy agreement specifies a wider recipient will be disabled.

3.4 Data access strategies

In the following, we present different data access strategies in order to help our approach to handle cases where non-functional properties hamper data access (volume, latency, etc.). Here, we present a data access model that describes what processings are required according to specific characteristics, affecting the way a client will connect to and download data from data sources. We identify different behaviors, where according to characteristic values, different data access policies could be adopted.

3.4.1 Preparation

First of all, and according to a user request, there is a need to identify which data source will provide the required data. In order to decide it, we rely on several elements:

— the data query issued by the user, which will define:

1. which concepts are required
2. which filter we want to apply to the data

— the data source characteristics, which defines whatever information are needed to access data
— the semantic information which associates concepts to physical fields, keys or properties (depending on data source type)

According to these elements, we identify the required data sources, and create a *graph* linking concepts from data source together.

3.4.2 Pre processing

Depending on data sources, there is a need for a preprocessing task, in order to define data access. When dealing with tabular data files, or RSS feeds, all data is retrieved once, and then processed. However, when dealing with database or Web services, we do not access the whole data stored in a data source.

In the database context, data will be accessed through a query, which has to be defined a priori. Relying on the elements listed in previous step, we create a database query, in the request format specified by the characteristics, sql, sqlite, mysql, etc.

When dealing with Web services, data is accessed by making service calls, sending inputs to services in order to retrieve data outputs. In this case, there is a need for several elements to prepare service calls:

— the data query issued by the user, which will define
— service description (WSDL, WADL, MSM for Linked Services, descriptor for RESTful services, etc.) which describes service operations
— semantic mapping which associates concepts to inputs and outputs (could be described in service description)

According to data elements, service callbacks are prepared, and will wait for data inputs before their execution.

Data access

Data access is the step where the data is actually retrieved, and each types of data source requires a specific technique.

First, plain data sources (e.g. TXT, CSV, XML) are all considered as a whole, and all the complete data set from this kind of data source is extracted once. During this step, data from feeds (RSS) and data from services that does not require output are also retrieved. Secondly, database sources are requested through generated queries.

Finally, according to the data already retrieved, service callbacks are created.

3.4.3 Adapt to physical characteristics

Several characteristics directly affect data access, technically. *Data volume* is defined as a discrete scale, as presented before. In case where the data source representation does not specify it, the default volume value is set to small. Technically, low volume sources can be accessed at any times, according to needs. Medium volume sources involve delays and high processing times, so a cache system should be setup. A high volume source can either not be directly queried in a synchronous way, because the volume of data it contains implies a too high response time. In this case, it is recommended to set up a synchronization system, where data is periodically retrieved from the source and saved in a local cache. In the case of big data sources, when data cannot be accessed directly because requests takes to much times, we recommend setting up big data mechanisms such as Map-Reduce to process data in addition to a local cache.

Latency represents the delay (in seconds and milliseconds) between a data request and the received answer, set by default to 0. The system statistically update data source latency value after each request. When latency is high, mechanisms of cache or pre-loading are set up.

Update period represents the delay between 2 major changes in the data source, set to daily by default. An update period variation will not influence small data sources, but from medium to big data sources, the cache and synchronization systems will be impacted. A short update period will force to increase synchronization delay, and cache will be cleared more often.

3.4.4 Direct and indirect access

We build our access strategies according to two different models: *push* and *pull* strategies.

In pull based strategies, there is no background workflow, data sources are requested directly and on demand. This solution does not imply any storage or synchronization, we request sources, combine results and return the response. This strategy is only available, with low or medium data sources (with or without cache).

In some cases where data source volume is high, or if data source is a stream, data

has to be retrieved in background, and stored at a given frequency. We call this a push strategy, it provide some interesting reduction of time and cost for request that involves big data transfer or processing, or in case where the request has a high demand certainty.

3.5 Conclusion

In this chapter, we focus on different challenges to propose a flexible solution that handle data source diversity. We propose a solution in order to improve data source management by focusing on data source themselves. We rely on a meta-model to provide a structured way to describe data source characteristics. With help from this formalism, we show how to adapt data access to the specificity and diversity of data sources. The data source meta-model, presenting data source *physical* characteristics, allows a dynamic and transparent management of data sources. It contains the necessary and sufficient information in order to provide a complete and independent data source access. On top of that, our meta-model allows to describe the data itself, gathering the necessary information about what this extracted data contains, this information will have to be populated by the system and the data sources themselves during execution. To summarize, we presented a set of models allowing to fully describe data and data sources, at different levels. This formalism describes the necessary information to automate the data source management.

Based on this information, we go further into our data access automation objective, by providing data access strategies, as a set of characteristic-based optimizations. These strategies allow to improve the data access, by focusing on metadata to setup the required mechanisms, to interact with data sources, in order to request and extract the necessary data. With help from these strategies and models, we build a generic client that fits with data source specificities. Our approach reduces the amount of hard coded information, and provides a dynamic data source management approach.

To conclude, our model provides a dynamic and transparent data source management, providing specific client adaptation. Whereas this model allows to automate the data source access, the scalability to a large number of data source cannot be guaranteed *a priori* by our model or strategies, and requires further research.

In the next chapter, we go further into our global integration objective. We rely on the models and strategies proposed to design a generic and adaptive software architec-

ture. The architecture we present in the following chapters will take benefits from our models and strategies to dynamically orchestrate the different processes that are necessary to perform a complete data source combination. This adapted task orchestration will provide a scalable solution to our global objective.

Chapter 4

Adaptive Workflow Architecture

4.1 Introduction

Over the past few years, big data has generated a lot of interest from researchers and industrials, especially for the benefits it produces in the field of data processing, and more precisely for data analysis, capture, storage, transfer and visualizations. Approaches and techniques answer the big data challenges, known as the four Vs: Volume, Variety, Velocity and Veracity. Defining our smart data architecture, we partially address these challenges. We address the Variety challenge, providing models and strategies to support the many types of data sources, not only the existing types but also those which are emerging by providing a meta-model to describe data sources. We provide a solution to the Velocity challenge, by giving to our architecture the ability to adapt its workflows to the different cases that may appear, optimizing resource orchestration to improve the response time. We address the Veracity challenges by providing resources and techniques to analyze, combine, repair malformed pieces and clean data from inconsistent pieces and duplicates. On top of that, we put more focus on data quality through the use of semantics, metadata and intelligent processing techniques.

The main difference between big data and smart data resides in the fact that big data produces more and more data, finding new correlation and patterns between data whereas smart data tends to extract the valuable parts from the data sets we own. In smart data approaches [Dustdar et al., 2012, Lopez and Kotoulas, 2012, Sheth, 2014], propositions focus on data quality rather than quantity, on building smarter architecture able to take advantage of linked data and open data projects, such as smart cities.

In this data-driven context, working with Web-standards and resources comes as a solution. The problem that appears while using HTTP and RESTful resources is that volume hampers data exchanges. Some approaches have addressed the data transfer issue in HTTP-based solutions relying on technical approaches such as protocol optimizations to improve the transfer time.

4.1.1 Motivation

Workflows have emerged as a powerful technology when trying to automate and orchestrate tasks (services, processes, etc.). Due to changes of context, laws and policies, but also due to advances in technologies, new methods, and practices, workflows are being continually changed. Having defined our smart data process, we proposed a static workflow, able to perform the desired combination task of two data sources, in optimal and predefined conditions.

However, in the previous chapter, we have seen that data source can present very specific characteristics, which can have a very strong impact on the data access on the one hand, and on data integration process on the other hand. There is a need to add dynamicity to this static workflow according to what data source characteristics are involved, in order to improve the different process, in terms of execution time, and quality of data, but also in order to avoid issues, such as inconsistencies, incoherence, and noise production. In our approach, we consider unrelated and irrelevant data as noise.

On top of that, in order to go further into the definition of our data integration platform, relying on the previously defined framework. There is a need for a software architecture able to handle this dynamic workflow adaptation, and in addition, able to perform task execution.

4.1.2 Challenges

In this chapter, we rely on our previous contribution, and propose to address this challenges:

— First of all, we propose an approach to adapt the global process workflow according to the data source characteristics, defined in the data source model

— Secondly, we propose a software architecture, based on the resource-oriented paradigm, able to support adaptation and execution of adapted workflow

In order to perform this, we rely on existing workflow adaptation techniques to build our proposal. Using data source models (for representation and access), we adapt tasks orchestration in order to fit with the data source characteristics. Having defined this workflow adaptation technique, we review classical software architecture design patterns in order to build a proposition of architecture. Finally, we show how our architecture can adapt to different context by propose an optimization to handle high data volumes.

4.2 Related work : Workflow adaptation

Workflow adaptation is a field of research where we rely on different factors to adapt orchestration and execution of software tasks, i.e, workflows. Adaptation is a core technology to support long-term business processes in heterogeneous and distributed environments. According to needs, workflow can be adapted in different ways, before execution, according to source and service specifications, during execution according to events or data instances, etc. In this section, we envision existing approaches that allow the creation of adaptive workflow, especially late modeling and ad hoc adaptation approaches.

The most common techniques we are interested in to adapt workflows are the following:

— Late modeling: workflows are not completely specified, some parts (i.e., sub-workflows) of the workflow are left open to be fulfilled at runtime relying on environment changes, but it has to be known in advance, to prepare adaptation

— Ad Hoc Adaptation: workflow is adapted during runtime, but it cannot be decided in advance, and it is difficult to know if the adapted workflow will be correct or not

Here we present a review of some relevant approaches.

4.2.1 Late Modeling Approaches

Late modeling approaches are most of the time, based on a semi-structure of pre-defined workflow, which can be adapted at runtime or before runtime to fit with context execution and/or data samples. These approaches are especially planned to work with classical web services, and are not really well adapted to work with RESTful resources. We can choose to adapt either the execution order, or the components to use, by using

some kind of placeholders in the initial workflow, that will be replaced by the necessary component (service, code block, etc.).

In *pockets of flexibility* [Sadiq et al., 2001], they propose a framework to build flexible workflows, based on the definition of a partially specified model, where the full specification is made at runtime, and may be unique to each context. The rely on a separation of workflows into a dynamic, adaptive and flexible process. The workflow changing process is made part of the global process itself, by introducing a core process and a set of pockets of flexibility, which will provide the perfect functionality to integrate inside the open workflow instance.

In [Adams et al., 2006], authors propose an adaptation technique for the YAWL language [van der Aalst and ter Hofstede, 2005], with an approach based on the use of worklets, which are placeholders or gaps to be filled at runtime with an implementation. This approach rely on more rigidity on workflow definition. This approach relies on a repertoire of possible applicable actions which is made available and maintained for each type of activity, which are used to create dynamic and flexible workflows, under of form of a work plan, which is not a precise definition of tasks, but looks like a guide which may be modified during execution depending on context. The approach will rely on context to perform final decisions at runtime. Every execution will generate a context-specific set of deviations, which can feed a knowledge base, to learn from each execution to improve future adaptations. This approach requires less granularity when modeling process, improve reusability of sub-processes and provide a fault tolerant approach.

In Move [Herrmann et al., 2000], Hermann et Al. proposed a semi-structured workflow model, in order to represent what they call *vagueness* (incompleteness of information). Their approach rely on the common concept of roles, activity, entity and the relation between them. The different users of the systems have roles, which gives them the ability to perform activities upon entities (which can be documents, tools, etc.). The dependence of each relation between these elements define the workflows, interpreted as diagrams Having defined this system architectural style, they defined the notion of vagueness as a lack of information, defined by either a user intended omission or a doubt. According to what happens, users can specify incomplete information or express doubts, which will help annotate the processes. The adaptability in this approach is made through the use of vague relations, which does not requires a definite specification, they are not necessary connected between two specific elements. The final decision will be made according to what happens in the system. This approach

is particularly efficient, with what they called socio-technical systems.

4.2.2 Ad Hoc Adaptations

In ad hoc adaptation, workflows are adapted during runtime when needed, most of the time according to context of execution. This cannot be planned or decided in advance, and it is difficult to know if the adapted workflow will be correct or not. The main challenges, is to provide approaches that will guarantee correct workflows, since it cannot be covered by late modeling.

Minor et Al. [Minor et al., 2007] propose an hybrid approach between late planning and ad-hoc approach to perform structural changes in process workflows. They proposes a workflow modeling language based on the classical control flow elements of workflow patterns [van der Aalst et al., 2003], and extended with several elements such as *placeholders* for sub-workflows, for sub-diagrams and breakpoints. They model these workflows as trees, where nodes represent elements (tasks, placeholders, ...). Each node encapsulate its execution logic. This approach propose a suspension mechanism which allows to modify workflow parts, while the remainder of the workflow is being executed. According to context, the ongoing (i.e., running) original workflow (called workflow definition) is adapted according to case based reasoning. The best matching case are applied to the workflow.

Dorn et Al.[Dorn et al., 2009] propose context-aware adaptive service mashups, where services will be classified by *capabilities*. The adaptation process rely on a monitoring system, which observes context changes in mashups. When mashups are affected, system performs a reevaluation of of requirements, then generates a set of services with the best fitting capabilities. They proposed an algorithm to help recommend services refinement (service are replaced by an equivalent with the same capacity) according to a set of rules. Services are compared, according to the function they cover, and a score is computed. Finally, services are selected amongst this ranked set, and mashups are reconfigured.

These approaches propose wider adaptation possibilities, and more flexibility, but they also requires mechanisms in order to constantly verify the correctness of the generated adaptations.

4.2.3 Conclusion

Since we planned to have minor changes on our workflows, they can be modeled in advance. However, this changes required information about execution context to be finitely decided. We rely on a late modeling approach, where execution order will be changed, or where sub-workflows will be added to the process (e.g., heterogeneity resolution before, during or after data combination) according to context execution, data characteristics, or data sources constraints.

4.3 Adaptive Workflows

In this section, we propose a workflow late modeling approach, based on the definition of a workflow, whose sub-workflows and tasks could be changed, duplicated, or switched according to context and data sources, in order to fasten the final core process execution. Our adaptive workflow technique relies on a late modeling approach, where we provide a generic complete workflow. This workflow will be modified at runtime according to the data source we manipulated.

4.3.1 Workflow representation

In order to represent workflows in our approach, we rely on a graph description representing workflows as directed graphs, where vertices are tasks (services callbacks or resource requests), and links being the dependency between these tasks. These dependencies will induce task execution order, and the orchestrator will rely on this description to perform message transmission between the different architecture components.

Figure 4.1 shows an example of workflow in this formalism. We introduce 3 services A, B, C and D. Processes are executed from left to right, after their dependencies. In our example, A and B have to be executed before C, and C before D. This basic workflow model will allow us to generate our task orchestration.

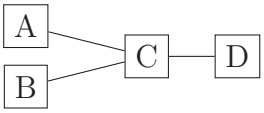


Figure 4.1 – Classical integration workflow

4.3.2 Standard procedure

As introduced before, our approach rely on a generic workflow, which represents the standard procedure to combine two generic data sources. We consider generic data sources as medium volume sources, with no latency, CSV format, no authentication, requestable easily. They are provided with a semantic mapping, providing one concept per column, and share a pivot concept. In this case, the process orchestration in order to integrate data coming from these two data sources called S_1 and S_2, is done as presented in Figure 4.2. Data is going through the following steps: download (Dl), decode (Dc), request for the mapping description (Sem), then both data set are integrated (I) and finally filtered (F).

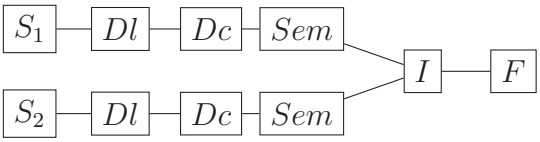


Figure 4.2 – Classical integration workflow

During execution data goes through different states, from the raw original format following data extraction, to our internal format that facilitates manipulation, and finally to the linked data format once annotated. The move from one state to another may have an impact on processing in terms of response time (especially when processing a huge data volume) or data consistency (streams VS static DB).

Therefore, the processing workflow can be envisioned as two connected workflows, where the connection point is the integration function. This way, we define two different parts in the integration workflow: **pre-integration** and **post-integration**. The *pre-integration* part represents the different functions applied to the data set from the extraction from the data source to the integration task. The *post-integration* part begins with data integration and ends with data rendering. This separation helps performing the tasks at different levels, first data preparation aims at preparing data for integration, then the integration task combines data from different sources, and then different functions such as filtering apply to the resulting set.

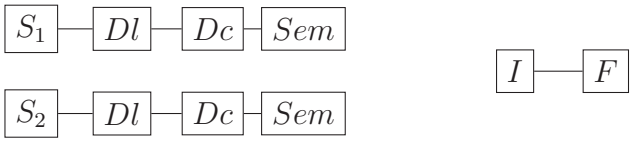


Figure 4.3 – Pre and post integration sub-workflows

4.3.3 Adaptations

According to characteristics, this standard workflow will be adapted to fit with the different use case that occur. There are different types of tasks that can be adapted. We identify different types of task adaptations: switch, duplication, delay, removal. According to our scenario use case, we define several adaptation rules.

Combination plan

The first type of rules, we propose is made according to the data we manipulate, and will define the source combination plan. According to our query, we identify a given number of data sources we have to integrate. According to the pivot field on which we will combine our data sources, we will identify in which order data sources are combined. We will then generate one *pre-integration* sub-workflow for each of these sources. In case where data sources does not require input, all the pre-integration workflow could be executed in parallel. It means no dependencies, task will be executed in sequence, but no matter how. The first one in the list will be executed, since no order is given, the one with the lowest id will be requested first. After that, the combination plan will decide which data source is combined with which. Figure 4.4 indicates how pre-integration workflows are executed in parallel (parralel in the workflow, sequenced in execution) and how combination plan affect the combination order.

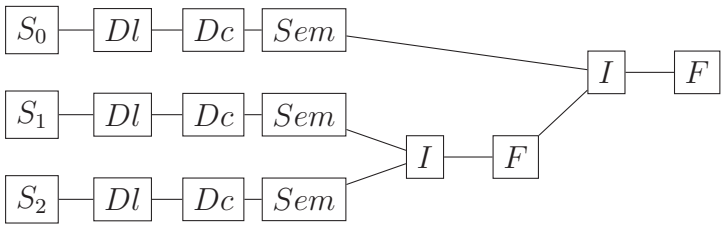


Figure 4.4 – Combination plan execution order

In some cases, data sources needs input in order to be requested. It is particularly true with databases, that requires a query, and some services that requires input. In this case, we will delay their pre-integration workflow, on order to wait for the necessary data inputs. Figure 4.5 shows an example of combination between a tabular file and a database. The data from the first source will be used to request the data from the second one.

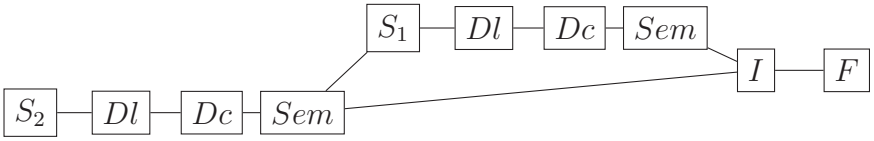


Figure 4.5 – Combination with input

Volume and latency optimizations

In order to optimize the workflow execution, in is necessary to perform workflow adaptation according to data sources characteristics.

The first task that will be necessary, according to cases, is the filtering task. It is helpful to duplicate or move the filtering between the different steps, in order to remove the malformed or unrelevant pieces of data that could slower the process. The filtering task can be placed after or/and before the integration process, so that data cleaning is performed once or twice. In this case, the filtering process can be placed before the integration task or before the semantic transformation task. Fig. 4.6 shows a workflow where a filtering task is inserted before the integration process.

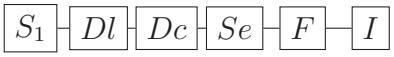


Figure 4.6 – Pre-integration workflow with earlier filtering

Typically, data is transformed into linked data before the integration task, because semantic annotations facilitate the integration process. When it comes to big data volume, combining two huge data sets can be tedious and time-consuming. With high volume data sources, if the scenario allows it (e.g. same schema), it can be very interesting to move some tasks forward, in order to reduce the volume we have to process. In some cases, process may be deleted from the pre-integration part and placed in the post-integration part. We propose an optimization of our workflow by swaping components in order to perform the combination of large data sets, on top of that, it may be interesting to perform a cleaning before integration. In the case where the data sources are databases with the same model, or CSV data files that have common fields, performing integration before semantic transformation optimizes the process, because it reduces the size of the data sets to annotate. Fig. 4.7 present an example of workflow, presenting an earlier integration, before the semantic transformation. In this case, we must be sure that the data allows the combination before the semantic annotation, it is possible when data share the same model, or belong to the same source.

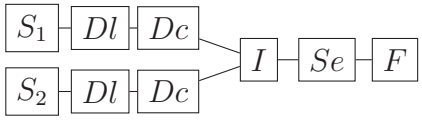


Figure 4.7 – Workflow presenting early integration

Finally, and as presented in the previous section, in some cases, specific tasks can be added to the workflow, especially when the semantic model is missing. In this case, a semantic tool allowing to recognize the semantic graph associated with a data source could be inserted before the semantic transformation task. In this thesis, we do not provide contributions about semantic recognition of data sources, but present several approach in related work, in order to help semantic annotation of sources. The semantic extraction/recognition tasks, can be any approach to retrieve semantic over sources, as long as it is exposed as a service or through an API. Fig. 4.8 shows an example of a classical integration process where one of the data source does not provide any semantic mapping definition.

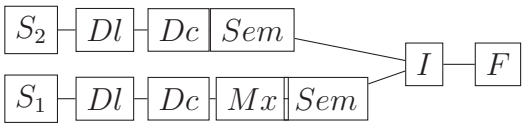


Figure 4.8 – Workflow with a semantic extraction process

This adaptive workflow technique, is implemented through a number of rules conditioning the organization of processes in the architecture we present in the next section.

4.4 Architecture of the solution

Based on this workflow approach, we design our adaptive resource-oriented architecture, based on a study of Web architecture design patterns.

4.4.1 Comparative study of existing architectural design patterns

In order to organize and structure the previously introduced tasks into a distributed architecture, we have studied different architecture design patterns, as summarized in the following. The first pattern is an Enterprise Integration Pattern, the next three patterns are related to Service Oriented Architecture, and the last one is related to

Resource-Oriented Architecture. We present these patterns and discuss their advantages and drawbacks.

Shared Databases

A shared database architecture [Hohpe and Woolf, 2003] is a enterprise integration pattern where different services and components share the same data storage. This type of architecture presents advantages related to the data storage, when enterprises need information to be shared rapidly and consistently. All services and components share the same schema, which helps interaction. Moreover, database studies have shown that this type of architecture is highly adaptable to big volumes, due to database characteristics. Database caching is also widely supported. Nevertheless, this architecture is completely centralized, preventing use of third party services, or components with heterogeneous schemas. Furthermore, the database as a single point of failure becomes an architectural limitation.

Classical SOA architectures

Classical SOA architectures [Erl, 2009] consist of a set of services and static workflows that are compositions of these services. A workflow describes service calls and explicit transformations of data flows. Each use case in the architecture requires a manually crafted and specific workflow. Using such an architecture allows gaining benefits from the principles of SOA, i.e. platform- and location-independent loosely coupled services. It allows to use different service types (SOAP, REST, ...) provided by a variety of third-parties. Despite these advantages, SOA architectures lack adaptivity due to the required hard-coded information. Each task has to be manually integrated into workflows, for example data mediation or caching. Moreover, querying components that are not deployed as services becomes a complex task and requires adapters.

Layered Architecture

A layered architecture [Erl, 2009] gathers services together in layers according to their functional similarity. Each service from a layer may only interact with services of the upper and lower layer. This pattern presents a good cohesion within layers, since groups of common featured services are gathered together. This cohesion brings

a good separation of concerns, making layers reusable and easy to maintain. Structuring services into layers limits coupling, which simplifies development and facilitates component replacement. Unfortunately, a common schema is needed in the architecture, otherwise it becomes necessary to insert transformation components between each layers. Furthermore, there is a lack of modularity due to this layered data exchange, it is impossible to swap services in workflows for optimization purpose, making the architecture inflexible. And finally, it is difficult to structure layers. If grouping conditions are too strict or too soft, the architecture ends up by having either one layer per service or a global layer which contains all the services.

Enterprise Service Bus

Enterprise service bus (ESB) [Chappell, 2004] is a type of architecture based on a message bus where various components connect to a service bus via their service interface. Service composition are managed through the architecture by creating routes. Routes describes service interactions, and a message broker handles the data flowing from and to components. Enterprise service bus architectures present all the advantages of SOAs, including service independence and loose coupling. The usage of a message bus simplifies the integration process. It provides a precise data management in service composition. However, Enterprise service buses have two main drawbacks. First of all, message routes are static, which forbids dynamic composition, and secondly, message bus needs service adapters to connect to different resources and components.

Resource-Oriented Architecture

In a resource-oriented architecture [Richardson and Ruby, 2007], all the software components must be resources with RESTful interfaces, which means they are accessible through their URI via HTTP methods (GET, POST, PUT, DELETE,...). A RESTful architecture must respect several principles [Fielding and Taylor, 2000], as follows:

— *Uniform interface* ensures that the method information is kept in the HTTP method (we use GET to retrieve a representation of a resource, POST to create a new resource, PUT to upload new representations and DELETE to delete resources), this property also helps to respect statelessness

	advantages	drawbacks
Shared Database	- Same schema - Big volumes - Cache	- Centralized - No third party - Database SPOF
Generic SOA	- Independence - Third parties - Service diversity	- Static workflows - Hard-coded config - Limited to services
Layered SOA	- Cohesive - Limited coupling - Good reuse - Easy maintenance	- Inflexible - Complexity
Enterprise Service Bus	- SOA advantages - Easy coupling - Good reuse	- Static routes - Service adapters
Resource-Oriented Architecture	- Independence - Uniform interface - Dynamic - Adaptive	- Learning curve - Resource adapters

Table 4.1 – Comparative table of architecture designs

— *Addressability* ensures that the information about the scope of a resource is kept in the URI, every object will have its own specific URI
— *Statelessness* means that each request happens in complete isolation, and the server does not store any state information, each request contains all the necessary information, thus improving scalability of the solution
— *Representation oriented* means that interactions with resources are made using representations of these resources, request header (such as accept) specifies the desired format

The REST architectural style provides advantages such as a complete independence of underlying platforms and languages, universal interface and access thanks to HTTP methods. Resources can be dynamically composed and reused to fulfill a request.

Table 4.1 summarizes the advantages and drawbacks of the different architectures presented above.

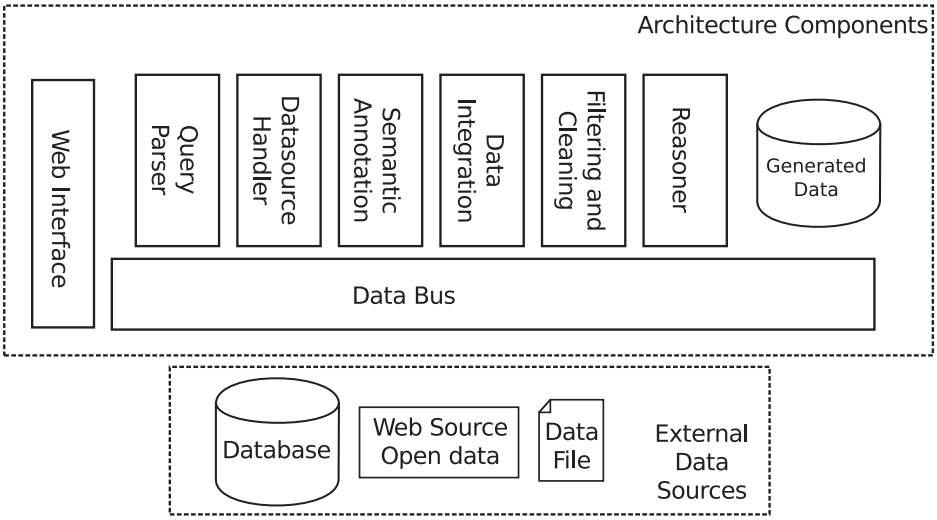


Figure 4.9 – Architecture Resources

4.4.2 Overview of our architecture

In order to organize and structure our tasks into a distributed architecture, we rely on the previous study to propose a resource-oriented architecture enhanced with a *data bus*. Our architecture handles components as RESTful resources available through a uniform interface via HTTP methods. It overcomes the drawbacks of ESBs, and presents a generic interface to components. In this section, we rely on REST and SOA principles to build a generic, scalable and modular architecture that will support our adaptive workflows.

We defined a set of architecture components, as shown in Fig. 4.9, exposed as generic RESTful resources identified by URIs and accessible through HTTP methods. These *business resources* are the core of the architecture, they handle the main data processing tasks. In our architecture, we rely on separation of concerns, to improve flexibility and reusability. We separate our framework into different components, each of them handling a particular type of task. In addition to these core resources, we present a set of resources that support task configuration and administration, referred to as *management resources*. According to scenario use cases, this architecture generate an adapted workflow, using the available components to fulfill the desired behavior (extraction, combination, filtering, etc.)

4.4.3 Architecture components

These components are the core resources that allows the architecture to work. We separate architecture resources and services according to what they handle. We call *core resources* the service that are used to perform the concrete task of the integration process, (data extraction, semantic enhancement, etc.)

In order to make architecture management and administration easier, and to avoid manual configurations as much as possible, we provide our architecture with management resources, accessible via a set of APIs. Through these APIs, users can alter resource behaviors and settings, add or remove data sources, request the generation of a mapping for these data sources, plug or unplug core components from the plugin registry, define specific business-oriented rules.

These resources are managed by our architecture, with help from the data bus, which we present here.

Data bus and orchestration

In order to handle different data flows between resource (resource requests), we define an orchestrator, which act as a data bus and receives HTTP requests from the Web interface. Each data sources, resource and necessary services are stored into a repository, to which the bus has access.

On request reception, this component extracts the SPARQL query from the request payload, and forward it to the request parser. Based on this data query, the query parser generates and returns an adaptive workflow, defined as a set of data sources and generic tasks with input and output.

According to this workflow, the orchestrator handles requests to the different architecture components, retrieving data responses and forging HTTP requests, according to query and data source characteristics. The orchestrator is defined as a RESTful resource, user queries are sent through HTTP requests.

```
GET /query?user_token=AS65G&query=SELECT+%3Femail_value+%3Fcampaign+WHERE+%7B%0D%0A++%3
    Femail... HTTP/1.1
Host: restful.alabs.io
Keep-Alive: timeout=15
Connection: Keep-Alive
```

Listing 4.1 – Sample of HTTP request embedding a SPARQL query

Query parser and reasoner

In order to interact with the system, we rely on data queries that are sent to the orchestrator. **Data queries** are formatted in SPARQL and involve semantic concepts. Listing 4.2 gives an example of query that involves a set of concepts belonging to our scenario.

```
PREFIX al: <http://restful.alabs.io/concepts/0.1/>
SELECT ?email_value ?campaign WHERE {
    ?email       a                    al:email ;
        al:has_email_value  ?email_value .
    ?email_value a                    al:email_value .
    ?clic        a                    al: clic  ;
        al: clic_email        ?email ;
        al:clic_campaign      ?campaign .
    ?campaign  a                      al:campaign .
}
```

Listing 4.2 – A data query example

Data queries are forwarded to the query parser, which extracts the corresponding algebra, as a set of subgraphs and concepts [8].

Our architecture will then take benefit from an engine acting as a simple **reasoner**, that will help making the process dynamic. With help from the query parser, we retrieve the data sources providing the data pieces whose semantic concepts fit with the concepts provided in the query. In this case, we look for concepts that are equals, or where source concepts are subsumed by request concepts. As an example, google identification address, being `@gmail.com` address, will fit with an email address concept. Doing so, we will be able to provide a set of data sources to combine in response to a specific query. This will provide the necessary semantics in order to help the workflow adaptation, involving different architecture resources and the data sources that are needed. This information will be retrieved thanks to a set of rules, that associates different behavior to each type of characteristics provided by the data sources involved. We use classical AI mechanisms and algorithms such as semantic Web inference engine (Jena, Pellet, Euler EYE, HermiT) [9].

8. See https://github.com/semsol/arc2/wiki for a documentation about the SPARQL parser we use.

9. See http://www.semantic-web-journal.net/sites/default/files/swj120_2.pdf for a comparison of reasoners.

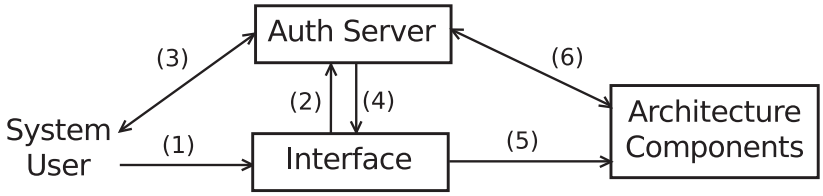


Figure 4.10 – Architecture authentication process

Authentication and data security

Our architecture interacts with each of these resources, which represent software components and data files. In order to enhance security, protect data and limit access to this resources, we overlay our architecture within an authentication layer, relying on the oAuth [Leiba, 2012] authentication framework.

OAuth relies on a authorization server which authenticates user access upon a resource from a server. The framework relies upon authentication tokens generated by a authentication server. These tokens can be used to access resources owned by a system. The decentralized pattern of our architecture forces us to authenticate each resource exchange.

The system user connects to the architecture interface (1), the architecture redirects him to the authentication server interface (2). The system user logs in through this interface (3). The authentication server generates a token (4) that authorizes the Web interface component to access the different resources in behalf of the user (5), each layer verifies with the authentication server the token freshness and validity (6). Figure 4.10 illustrates the authentication process. We rely on the Zend PHP oAuth component to implement our authentication component.

4.4.4 Architecture Resources

In the following, we present the *core* resources that handle the necessary tasks of the integration. We also present the *management* resources forming the configuration API, that allow to configure each component usage in the background. Please note that all components do not necessarily have a resource configuration API.

Data Source Handler

The **data source handler** allows to extract data from the different data sources involved in the query subgraphs. This resource accesses each data source and extracts data with the help of the data source description (see Section 3.3).

The architecture benefits from a set of management resources for data source handling, allowing to perform the four CRUD (Create, Read, Update and Delete) operations to manage data sources. This resources provide an API that allows to perform the four CRUD operations (Create, Read, Update and Delete) over data sources. Data sources are retrieved and manipulated with their *ids* and according to the model instance that describes their characteristics. As an example, a GET request over the data source configuration resource with the `id` of a data source: `GET /datasource/U1` returns a JSON object representing the data source with the `id` $U1$.

Listing 4.3 illustrates an example of data source configuration resource.

```
{
    "id": "U1",
    "format":{
        "syntax":"mongodb",
        "schema":{
            "endpoint":"http://153.75.28.26/schema_def",
            "syntax":"JSON",
            "structure":"JSON-S"
         }
    },
    "uri":"mongodb://153.75.28.26:8080/myDBendpoint",
    "username":"user1",
    "password":"76ls6h",
    "databasename":"maindb"
}
```

Listing 4.3 – A data source configuration file example

Relying on this information, the resource retrieves (or deploys, according to access strategies defined in Section 3.4) an adaptive client that handles the characteristics of the data source, authentication, format, volume, etc.

Semantic Annotation Resource

Relying on the semantic information provided in the data source representation, the **semantic annotation** resource will either annotate with concepts or transform data into linked data.

The resource uses different techniques to enhance semantics of raw data, depending

on the kind of data source. As an example, for CSV file sources, we rely on the RDF123 approach [Han et al., 2008] to transform raw data into linked data. This approach relies on expressions to map the contents of spreadsheet columns to linked data.

Depending of data sources, the semantic mechanisms can vary. Our generic meta-model and our decentralized architecture allows to rely on any kind of remotely available semantic enhancement approach. As an example, we rely on the open-source RDF123 approach [Han et al., 2008] to semantically annotate tabular data files. This approach is based on a column-based mapping, in which we defined expressions that helps to translate spreadsheet data to linked data. The architecture stores these expressions into RDF resources, called map files, and associates these map files with data sources. In the same way, semantic enhancement of database sources relies on the D2R approach [Bizer, 2004], in order to transform data from relational databases into linked data. The D2R platform is based on RDF mappings that attach conceptual graphs to SQL requests, giving access to relational data through a SPARQL endpoint.

Relying on the semantic information provided in the data source representation, the **semantic annotation** resource will either annotate with concepts or transform data into linked data. In order to manage semantics for each data source, we provide our architecture with management resources to create, modify, and delete mapping files and semantic mapping information. Mapping file generation relies on existing third party approaches for semantic annotation and transformation, depending on the type of document or data source we want to annotate or transform into a semantically explicit representation.

As an example, the mapping information for a data source can be retrieved by a GET request over the semantic resource with the `id` of this data source: `GET /semantics/#id` returns the required mapping file in order to add semantics to data extracted from data source `#id`. A POST request at the same URI is used to submit a new mapping file, DELETE and PUT to destroy and update. Depending on the data sources, the mapping contains different sets of rules, and the URI of the semantic transformation or annotation service. In order to process the data source to extract the corresponding mapping, when available, the following POST request: `GET /semantics/extract/\#1` will to generate and return the mapping file.

Data Integration

The **data integration** resources interconnects the data sets that have been extracted from sources and annotated with linked data concepts. The integration resource aligns the different concepts, relying on the user data query to construct the graph represented by this query. The architecture analyses the concepts in the query and prepares data for the merging process, detecting the pivot values if they exist, relying on metadata to connect data from the sources involved. The data combination mechanism will then perform a scalar product on field sets according to pivot fields, in the same way that a join would be performed.

To detect possible heterogeneity problems, could they be syntactic or structural, and to reconcile them, we rely on our previous DMaaS approach [Mrissa et al., 2013]. This approach analyzes semantically described data, using a decentralized (peer to peer) repository of mediation services, which are Web services dedicated to data conversion. The DMaaS approach classifies data heterogeneity issues according to the syntactic, structural and semantic levels, and provides adapted mediation along these levels.

This approach rely on service descriptions, analyzing concepts of input and outputs, to detect heterogeneities, proposing workflow adaptation to handle their reconciliation. We adapted this conflict detection mechanism so that it is able to work with our data model instances. Data source models and concepts are analyzed together with query metadata, and workflow is adapted on heterogeneity discovery. We intercept data requests and responses to perform the data transformation.

Cleaning and Filtering Rules

In our architecture, when data is processed in one resource or another, noise and inconsistencies may appear, as well as duplicate values or instances. The cleaning and filtering resource handles different data cleaning tasks, including data duplicate removal, incomplete data removal, formatting and encoding issue processing and data removal when an issue cannot be solved (damaged data).

In addition to these resources, the user has the possibility to provide specific filtering rules, to ignore specific data values, or to limit fields to range domains. Therefore, the cleaning and filtering resource has a management API, where users can manage their specific rules. Listing 4.4 presents filtering rules samples.

```
{
```

```
   "@context":{
       "@vocab": "http://example.com/filter/",
       "vcard": "http://www.w3.org/TR/vcard-rdf/",},
   {   "@id": "http://example.com/filter/r1",
       "data": "vcard:email", "operator": "not-contain",
       "value": ".org"    },
   {   "@id": "http://example.com/filter/r2",
       "data": "vcard:age", "operator": "more-than",
       "value": "20"      }
}
```

Listing 4.4 – Filtering rules example

In addition to cleaning and filtering resources, we provide a management resource allowing users to publish their own cleaning rules. These specific rules allow to ignore specific data values, or to limit the range domain of a concept. We define an API to publish, search and remove these rules as follows: The URI `/filter` allows to create, retrieve, delete and update rules according to their id, an example of rule creation is shown below. In addition, `/filters` allows to retrieve a set of rules for a specified data source id, through GET requests. Listing 4.5 shows an example of HTTP POST request that publishes a filtering rule.

```
POST /filter HTTP/1.1
Content-type:application/x-www-form-urlencoded;charset=utf-8
Host: restful.alabs.io
Content-length:200
id=r1&data=vcard:email&op=not-contain&val=.org
```

Listing 4.5 – Sample of HTTP request to publish a filtering rules

4.4.5 Use case illustration

Figure 4.11 illustrates the data that flows during a query execution. The example showed in this figure presents the following data exchanges

1. firstly, the system user creates a query through the Web interface
2. the query is send through the data bus of the architecture to the query parser
3. the query parser explodes the query and extract relevant concepts and relations ($foaf : user$ has an $foaf : email$, $foaf : user$ open $domain : campaign$, etc...)
4. according to concepts from the query, and data source models, a set of relevant data source is selected
5. data is extracted from the sources according to query information, and then transformed into a pivot format for manipulation (JSON or JSON-LD)

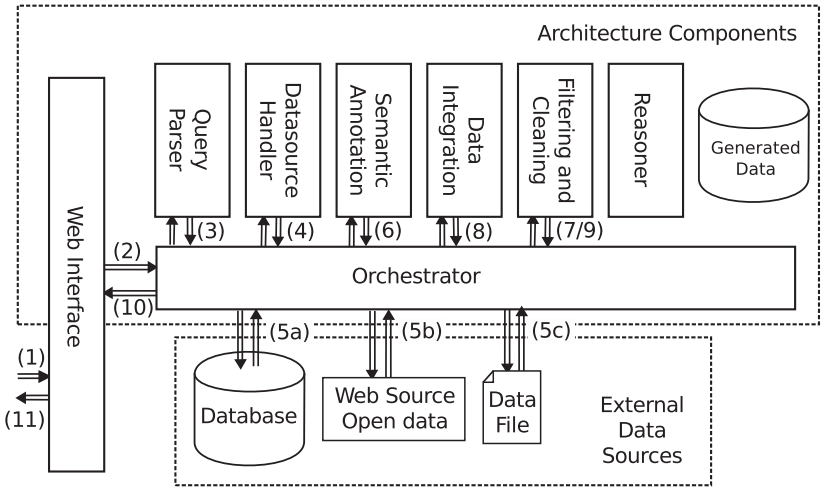


Figure 4.11 – Use Case Data Flow Representation

6. data is semantically enhanced
7. data is filtered one time (to cleanup eventual and semantic annotation problems), in the case where the query (or the user) has provided filters, some piece of data could also be removed
8. data sets are combined according to their pivot values (heterogeneities are resolved at runtime)
9. data is filtered a last time (to cleanup unresolved combination issues)
10. data is send to the user through the interface, directly under the form of a JSON-LD data set, or via an URI giving access to the result resource

4.5 Evaluation

In this section, we evaluate our different models and the resource-oriented architecture introduced previously in chapter 3 and 4. We rely on an implementation of our Architecture for Integration of Multi-Origin Data Sources (ArchIMODS) to evaluate the transparent and dynamic data source management and processing.

We answer the scalability challenge by evaluating the evolution of request response time over a growing number of data sources. We evaluate our architecture in terms of performance (response time) when answering a set of complex semantic queries over multiple data sources. We regularly increase the number of data sources and measure the response time.

Relying on our scenario presented above, we create two requests, involving subgraphs containing four concepts. We populate our scenario with a set of data sources covering the different subgraphs. Query 4.6 involves four subgraphs, implying data sources with different characteristics such as high volume (big database in our scenario) and privacy sensitive information (linked service in our scenario).

```
PREFIX al: <http://restful.alabs.io/concepts/0.1/>
SELECT ?email_value ?campaign WHERE {
    ?email       a                    al:email ;
        al:has_email_value  ?email_value .
    ?email_value a                    al:email_value .
    ?clic        a                    al:clic  ;
        al:clic_email       ?email ;
        al:clic_campaign    ?campaign .
    ?campaign    a                    al:campaign .
}
```

Listing 4.6 – Query 1 involving four concepts

Query 4.7 involves only three subgraphs, but includes user specific filters. This query introduces freshness sensitive data sources (streams in our scenario).

```
PREFIX al: <http://restful.alabs.io/concepts/0.1/>
PREFIX xsd: <http://www.w3.org/TR/xmlschema-2/>
SELECT ?email_value ?campaign WHERE {
    ?email       a                    al:email ;
        al:has_email_value  ?email_value ;
        al:blacklist_status ?status .
    ?clic        a                    al:clic  ;
        al:clic_email       ?email ;
        al:clic_date        ?date .
    FILTER (?status != 1 && ?date >= "1411477450"^^xsd:date)
}
```

Listing 4.7 – Query 2 introducing user specific filters

Fig. 4.12 and 4.13 shows the evolution of our architecture response time, when the number of data source grows. The graph also presents different composition techniques, that clearly shows the importance of adaptive composition. Workflow $WF1$ composes the steps of integration in a static way, which is quite well adapted for small data source sets, but does not scale when data source number grows. The second workflow $WF2$ introduces a dynamic composition, where the architecture is provided with the possibility to permute components, performing the filtering process before combining data.

This graphs shows that our architecture can handle the growth of data source number, as long as we use a dynamic composition model. In the case of the first

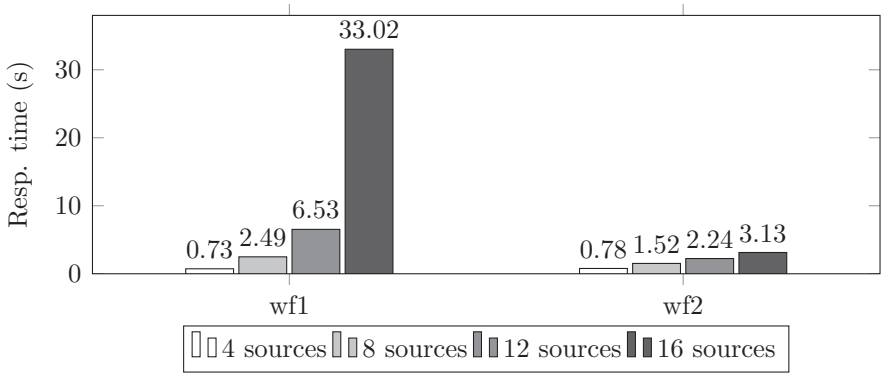


Figure 4.12 – Average response time for Query 1

composition model $wf1$ the combination of data become a time-consuming process, as response time grows exponentially. For more than 20 data sources, with the first workflow, architecture takes minutes to answer the query. With a dynamic composition workflow, avoid composing duplicates and non well formed data severely improves the process.

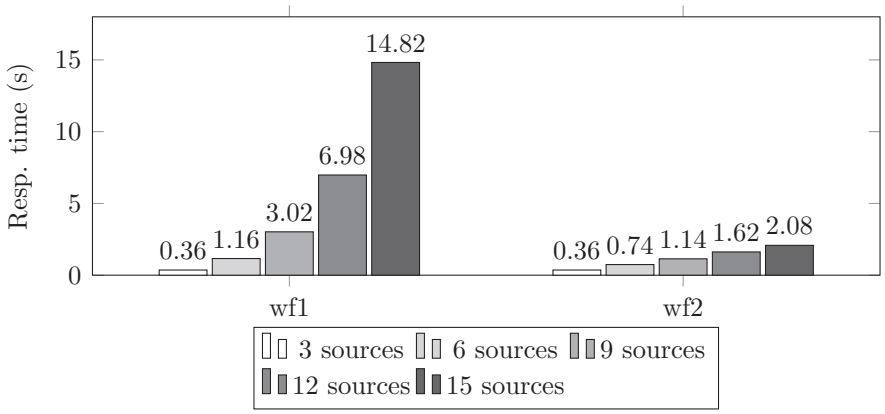


Figure 4.13 – Average response time for Query 2

The second query involves less concepts, and allows the architecture to give better responses with the first workflow, but it still takes more than a minute to answer Query 1 with 20 data sources. The second workflow adapts to the request and provides good results.

In this section, we demonstrate the scalability of our approach over a growing number of data sources. We provide performance tests, when answering complex semantic queries over multiple data sources. In the following, we demonstrate the adaptivity of our approach, by proposing an architecture optimization, to limit the quantity of data transfered over the network.

4.6 Architecture optimization

In the previous section, we rely on our data source models to design different processing strategies that adapt to the variety of data source characteristics, providing when needed: access models for volume value scale, cache setup for high latency or low update frequency, etc. Based on these data processing strategies, we build our adaptive architecture, which generates different processing workflows in response to a user request, in order to complete the required processing tasks. Our architecture optimizes and adapts workflows to handle the variety of data sources involved in the query.

4.6.1 The problem with HTTP and data Volume

Despite these models and strategies, data quantity still hampers data exchanges and remains a bottleneck in our architecture, especially when it comes to data transfer between resources or to render the request result to the user. By taking advantage of the many benefits provided by the REST architectural style and the use of resources, we unfortunately suffer from the limitations of Web protocols, and it is sometimes difficult to transfer large quantities of data through HTTP. This issue makes architecture decentralization difficult to realize, and our RESTful architecture suffers from a lack of responsiveness.

There is a need for an approach that minimizes data transfer and performs data processing tasks closer to the data source to reduce network traffic. Doing so, our approach could benefits from the advantages of the Web and REST principles.

4.6.2 Related Work on HTTP and data volumes

There is one common question when studying distributed architecture in order to manipulate big data volumes. The main issue stands in how to handle data transfer latency in SOA. Some approaches were proposed in order to solve or to dodge this issue.

Devresse et Al. [Devresse and Furano, 2014] propose an approach for adapting the HTTP generic protocol to improve its data access performance in data analysis applications. They created a library called *libdavix* allowing high performance computing world benefit from HTTP and the RESTful principles. This approach is focused at

the protocol level, and based on a dynamic connection pool coupled with the usage of the HTTP Keep-Alive feature, in order to maximize the reutilization of TCP connections. By avoiding useless protocol handshakes, reconnections and redirections, their approach improves efficiency of large data transport through HTTP.

In *Fast Optimised REST (FOREST)* [Ko et al., 2012], Ko et Al. propose a non-invasive technique which enables RESTful services to overcomes the traditional bottleneck experienced during transfer of large set of data. Their approach relies on encapsulating the original TCP data in UDP-based data transfer [Grossman et al., 2005] payloads. The approach allows to transfer data over UDP protocol between REST services in an orchestration. Even if evaluations shows good results, the approach does not seem to provide a real solution, it is a low level fixing to benefit from advantages of other protocols. According to the data volume challenge, they make the statement, which is also ours, that both data and data analytics need to be closely located to reduce the unnecessary network traffic, to improve efficiency. They introduced challenges and perspectives and present some approaches relying on services to perform big data tasks, but does not provide a solution to the volume issue.

Zheng et al. [Zheng et al., 2013] provide an overview of service-generated big data as well as big data-as-a-service, a flexible infrastructure providing common big data management functionalities. Their approach rely on cloud computing techniques to handle collection, storing, processing and visualization of data and they address some significant challenges, particularly about variety or volume and how infrastructure must support (and adapts) this variety and volume to provide fast data access and processing.

Van Der Pol et Al. [van der Pol et al., 2012] propose an approach based on multiple paths TCP [Ford et al., 2011] to transfer huge data sets over networks. Their approach relies on load balancing transfer through the different available paths relying on parallel multiple TCP requests. Their approach can handle different paths with different bandwidths, balanced over the different interface offered by the system. They propose a prototype

According to these approaches, it becomes clear that the most powerful solution is to minimize data transfers, process data volumes closer from the source and transferring data reference instead of data itself. Doing this way, we extract data and process it at the same time, only transferring data when necessary. In the next section, we present our resource-oriented architecture, the models that helps to build it, and finally, we present our solution to handle data volume diversity in our smart data

architecture.

4.6.3 Handling Volume Diversity

In order to handle volume diversity in our RESTful architecture, and to maintain a good response time performance, we propose a solution which reduces data transfer to a required minimum. We identify the different kind of data transfer and propose an approach that allows to handle the different processing tasks without having to forward the business data between the different resources.

Upon query request from the user, query is parsed and a corresponding algebra is extracted, containing concepts and subgraphs [10]. This algebra helps to detect which data source is involved in the extraction and combination process. Architecture optimizes the workflow, according to the different resources involved to handle the required processing tasks and a set of data sources which contains the data to be processed. This workflow is executed by the architecture, the data bus handles communication with resources, generating HTTP requests and retrieving responses from resources.

We identified different data transfers during the process
- data extraction from data sources
- data transfer to a task resource for a process (filtering, combination, semantic annotation, etc)
- data download (rendering to user)

In this context, data extraction and data download cannot be avoided, they are completely required. In order to process other tasks of the process, we design our RESTful resources so that they only manipulate the metadata instead of proper data. Each resource API is given the current metadata about the query and the data sets. Computing this metadata, remote processing codes and configuration are generated in order to complete the process managed by this resource.

As an example, we follow the complete workflow of a user request. The architecture receives a user request containing a query, which is forwarded to the query parser, which extract a grammar and identify the different concepts involved in this query. According to the concepts retrieved, architecture selects the corresponding data sources. Data source handler connects to these data sources and extract the business data (these are unavoidable data transfers). In order to annotate extracted data sets with semantic

10. See https://github.com/semsol/arc2/wiki for a documentation about the SPARQL parser we use.

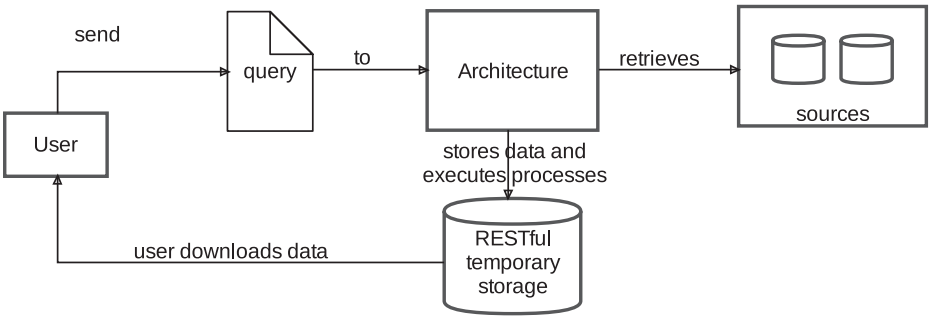


Figure 4.14 – Optimized approach to handle data storage and processing

concepts, metadata about data sets and query is send to the semantic annotation API which returns the semantic mapping, in order to attach concepts to the different fields of the data sets. In order to perform the data sets combination, metadata is sent to the combination API which analyzes the different schemas and semantic annotations in order to generate a combining process. For each step that requires a filtering process (as specified in the generated workflow), filtering API generates a resource containing the cleaning and filtering configuration. These configurations and metadata will help data the storage to execute the necessary steps close to the data, minimizing execution time by lowering latency and network time.

Storing data

In order to *temporarily* store data, our proposal is based on temporary data storage units, generated at each user request. Storage units act as file hosting services, they are generated at runtime, for each *new* user request, for a limited amount of time and contains the data and metadata for this request. We provide these storage units, with generic and configurable processing features, which can execute processing tasks over data. These units can execute decoding, combination and filtering above the data it contains.

Storage units are erased after a certain amount time, which has been fixed to a default value of 24 hours. This delay is customizable for each request. Time counter is reset each time a user reissues the query or reaccesses the storage. Storage units are accessible as RESTful resources, for management purpose or to retrieve query responses when data processing is over. When the processing tasks are over, the storage URI is given to the user, so that he could download the data sets answering to his request.

In order to handle the different tasks required to complete the different processes, we provide the storages with a functional engine capable of executing processing tasks directly above the data instances. The tasks resources of the architecture, will generate the configuration for each tasks, which will be transfered instead of data. As introduced before, temporary storages implements decoding, combination and filtering tasks, configurable via an API.

We provide data store with different processing engines, each representing an environment or an engine. Each data store is provided with an API, which allows its management, but also to register engine libraries or plugins, required to process the different languages or functionalities

4.6.4 Evaluation of our architecture optimization

In order to evaluate the scalability of our architecture, and how this optimization affect response time, we evaluate request response time when answering to a set of complex semantic queries over multiple data sources. We vary the number of data sources and measure response times.

Relying on the scenario presented above, we create two requests, involving four concepts subgraphs. We populate our scenario with a set of data sources, covering the different subgraphs. Query 4.8 involves four subgraphs, implying data sources with different characteristics such as high volume (scenario's big database) and privacy sensitive information (scenario linked service). This query also presents user specific filters.

```
PREFIX al: <http://purl.org/dc/elements/1.1/>
SELECT ?email_value ?campaign WHERE {
    ?email       a                    al:email ;
        al:has_email_value   ?email_value ;
        al: blacklist_status  ?status  .
    ? clic         a                    al: clic  ;
        al: clic_email          ?email ;
        al: clic_date           ?date .
    FILTER (?status != 1 &&
            ?date >= "1411477450"^^xsd:date)
```

Listing 4.8 – Query 1.1 involving the different concepts

Query 4.9 involves only a few number of concepts, and present less data manipulations. This query shows that latency is more due to data transfer than data manipulation.

```
PREFIX al: <http://restful.alabs.io/concepts/0.1/>
SELECT ?email_value WHERE {
    ?email      a                       al:email ;
        al:has_email_value   ?email_value ;
        al: blacklist_status  ?status .
    FILTER (?status != 1)
}
```

Listing 4.9 – Query 1.2 introducing a user specific filters

These results present two facets, with distinct results. In the first solution, which is the classical one, data is transfered between resources in a classical REST architecture. The second solution introduces our optimized proposal, involving localized data storage, with data manipulation processed directly upon data. Fig. 4.15 and Fig. 4.16 present our architecture response time evolution, when the number of data source grow respectively for query 4.8 and 4.9.

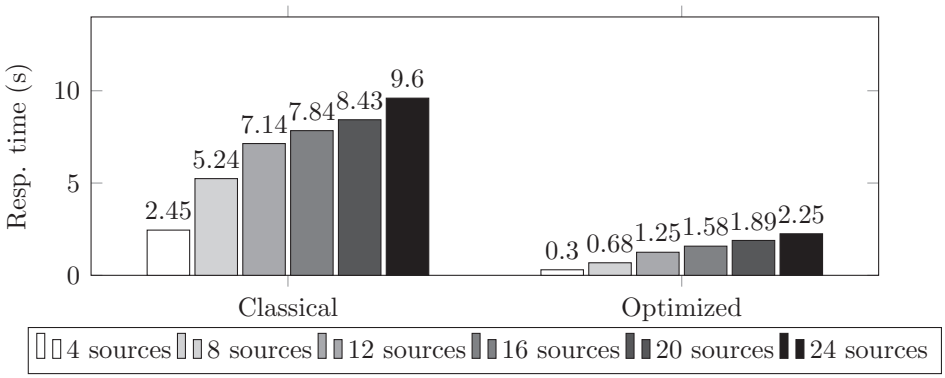


Figure 4.15 – Evaluation of average response time for query 1.1

This graphs shows that our architecture can handle the growth in data source number, as long as we use our optimized data solution.

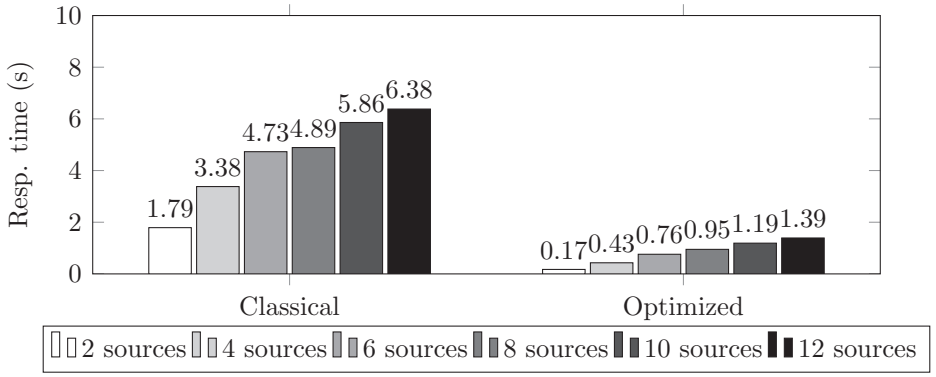


Figure 4.16 – Evaluation of average response time for query 1.2

The second query involves less concepts, and as a result less data manipulations, but our architecture still suffer from a very high latency due to data transfer. The second solution adapts to the request and provides good results, even when data grows.

In both queries, the classical approach suffers from a very high latency, even for a small number of data sources. Data is forwarded from resource to resource, where the tasks are processed. In our approach, the architecture exchanges only *meta-information*, including queries and metadata, with APIs and resources, and the size of this data transfer is much smaller than data itself. Data processing tasks are executed directly on-site, with the information gathered from APIs and tasks resources.

4.6.5 Discussing our optimization

In this section, we proposed an architecture optimization to focus on the latency problems that appear when dealing with diverse data volumes especially when transferring data between the different architecture components.

All along the smart data construction process, we rely on a temporary data storage, deployed as a resource with an URI, where business data is stored. Our architecture handles the communication between the different resources, by transferring metadata about the query and the data storage URI. Resources generate adapted remote processing codes, which are forwarded to storage units and executed on-site by the data storage engine. Therefore, data manipulation is performed on-site, and the data does not flow through the architecture. By reducing latency due to data transfers, we alleviate our decentralized and distributed architecture from the burden of data transfer, and improve the responsiveness of data-driven resource workflows.

4.7 Conclusion

There is an increasing need for building solutions to handle the different processes required to combine and semantically annotate data coming from heterogeneous data sources and generate smart data.

In this chapter, we rely on our previously introduced models and adaptive data processing strategies to propose an adaptive architecture to improve smart data management when data comes from different sources with heterogeneity issues, malformed data and duplicates. We propose a flexible solution to handle data according to data source characteristics, allowing the use of different data access and processing strate-

gies. To handle data management, we define a responsive architecture that orchestrates RESTful resources, accessible through their uniform interface to enhance interoperability. Our architecture allows converting and semantically annotating multi-origin data sources in order to produce a smart data set. It aims at being as generic as possible, independent from data sources, and adaptable to any use case.

We demonstrate the applicability of our architecture in the context of a scenario that answers the needs of our partner company. We also show its genericity by proposing an optimization in order to handle large volume.

In the next section, we focus on the quality of data objectives. We assume that uncertainties are everywhere on the Web, but is most of the time avoided by people, by making arbitrary decisions about contradictory information. We want to propose a model to represent this uncertainty, and techniques to handle this uncertainty, in our data integration concerns.

Chapter 5

Composition of Uncertain Web Resources

5.1 Introduction

Nowadays, modern information systems allow individuals, connected objects and organizations, to produce and publish a huge amount of data on the Web through Web APIs and public endpoints [Maleshkova et al., 2010]. Data is then collected from different sources and combined in mashups [Benslimane et al., 2008] to produce added value. However, the integration process raises several problems, as data may come from heterogeneous, contradictory or incomplete sources [Halevy et al., 2006].

Data uncertainty occurs when different data sources provide contradictory information about the same entity. In this case, there is a chance that each data source provides different information which we cannot solve by characterizing that one fact is right and the other is wrong. Information may be correct under some circumstances, and incorrect under others. Examining the origin of uncertainty - such as slight differences in physical event measuring, or differences in information provided by different data sources describing the same entity - may help understand its nature. However, the major challenge of today's Web is to provide a solution to deal with this uncertainty.

Instead of choosing a unique yet arbitrary version of information, we believe users should be given the whole spectrum of possibilities to describe an entity. The current state of the Web gives users the ability to select one single representation for an entity. As an example, when trying to find information about a book, the classical approach is to search the book name in a search engine, and browse the proposed Web pages until

the answer is found. This approach can be considered as a series of choices between available links, which will lead to the most appropriate representation of an entity (e.g. a book). When issuing a query to a Web API, the selected approach is the same, the mashup process typically discards data identified as incorrect during the integration process to provide the user with what is assumed to be the correct information only.

5.1.1 Motivation

Today's Web only gives users the opportunity to select one single representation from a set of representations of an entity. As an example, let us consider a user trying to answer the following query: *What is the date and city of birth of the author of the book "Les Miserables" ?*. Typically, while browsing through Web pages, a user will type the book name into a search engine, then he chooses one of the link proposed in the result page, and continues his navigation from this point to the answer. Such method only provides one certain representation at a time, the choice of what representation to select is left to the user who clicks on the links deemed to be the most appropriate.

As well, let us consider the same query issued to a traditional Web API or SPARQL endpoint. The mashup process typically discards data identified as incorrect during the integration process to provide the user with what is assumed to be the correct information only.

While the main objective is to allow the Web to deal with uncertain data, which requires providing a theoretical framework that allows for describing, manipulating, and exposing uncertain data to users, we identify several challenges in this context. First, Web resources should be able to provide access to multiple, simultaneous representations that leave the possibility to describe the same entity in different ways: there is a need to design and develop **uncertain Web resources**. Second, Web browsing and data mashups should deal with both uncertain and certain Web resources. There is a need to provide a solution to combine pieces of information from uncertain Web resources to answer user queries.

5.1.2 Contribution

In order to address these challenges, we propose a theoretical definition of the notion of uncertain Web resource and propose an interpretation model to access their uncertain representations. Then, we propose an algebra to evaluate this uncertainty in

the context of classical hypertext navigation that allows to combine data from several uncertain Web resources. We rely on these models to develop a set of specific operators and algorithms to evaluate uncertain data queries.

5.2 Data Uncertainty: State of the Art

Uncertainties have been processed in different contexts. We envision different approaches, handling uncertainty historically in databases, and more recently in services oriented application. Unfortunately, none of these approach handles the uncertainty that can appear when manipulating resources or when dealing with Restful applications.

5.2.1 Uncertainty in databases

Dong et Al. [Dong et al., 2007] propose a database schema matching approach based on uncertainty. Approach rely on generating an approximate matching between two schema. Their approach relies on constructing a probabilistic model to represent the data uncertainty that will help to decide in making a mediated schema, and later to reformulate queries for the different data sources. The mapping created rely on a select, project and join approach, which restricts the queries to single table on both sides, making each table mappings independent. They called these mappings *attribute correspondences*. In order to generate this probabilistic mapping, they propose two semantics, by-table and by-tuple. In by-table semantic, they generate the different query reformulations, considering the possible matchings (attribute correspondences). Probability of each reformulation is the product of each matching probability *used* in this reformulation. In by-tuple semantic, we have to generate certain anwser for every mapping sequence generated, in order to obtain a consequent result set. The complexity of by-tuple semantic is higher than by-table semantic.

Fagin et Al. [Fagin et al., 2011] envision data exchanges in presence of uncertain data coming from probabilistic databases. Their approach is a generalization of Dong et Al. by-table semantics [Dong et al., 2007] in which probabilistic matching are generated between tables in order to align fields. The generated results are associated with a probability value. Their probabilistic match approach relies on creating an arbitrary binary relationship between two countable (finite or countably infinite) probability spaces. This target must be countable probability spaces in order to create the map-

ping. Once this mappings has been done, they compute a matching degree in order to characterize the result of the query.

Agrawal et Al. [Agrawal et al., 2010] propose a local-as-view data integration approach dealing with sources containing uncertain data. Their approach rely on the concept of containment, making the following assumption: when integrating two uncertain databases, if a mediated uncertain database exists, this database must *contain* both databases.

Cheng et Al. [Cheng et al., 2012] propose an approach to evaluate probabilistic queries while dealing with uncertain matching of database schemas. Authors propose an approach based on the fact that concept values from both source and target schemas are often overlapped. Rely on this overlapping, they choose the correspondences which has the highest score and ignore the rest. They rely on these information to reduce the set of generated possible matchings to its minimum.

These works are strong although complex approaches to handle with uncertain/probabilistic data, these approaches has inspired our definition of uncertain Web resource. However, if it applies very well to database, these approach does not fit well when working with Web resources. One solution could be to layer data sources with a database endpoint, but it could not provide a sufficient solution for considering our composition semantics. These approach has lead our definition of uncertain Web resources.

5.2.2 Uncertainties on the Web

Several approach have been proposed to deal with uncertainty in other contexts than databases, most of the time in order to propose heterogeneous data integration approach.

Lamine Ba et Al. [Ba et al., 2014] propose an approach for data integration, combining data from web sources containing uncertainty and dependencies. Their approach confront and merge diverse information about a same subject from diverse web sources. They model the following data as probabilistic XML to process decisions [Kimelfeld and Senellart, 2013].

Sarma et. Al. [Das Sarma et al., 2008] envision what they call pay-as-you-go integration systems, which is related to our smart data architecture. Their system rely on a single point interface to a set of data sources, integration of data being made by creating a mediated schema for the domain.

Pivert et. Al. [Pivert and Prade, 2014] propose a solution to integrate multiple heterogeneous and autonomous information sources, resolving factual inconsistencies by analyzing the existence of suspects answers in both data sets. Their approach verify the data provided by two data source they want to integrate, if a data piece in second source invalidates a data piece in the first data source, it is considered as a suspect answer. Their approach finally return all the candidate answers to a query, rank-ordered according to their level of reliability.

Finally, *Amdouni et Al.* [Amdouni et al., 2014] propose an approach to handle the uncertainty of the data returned by data services, which they call uncertain data service. They define uncertainty at three levels, in the context of DaaS services, modelization, invocation and composition. First of all, they extend the Web services standards to model uncertainty of a service in its own description. This model introduces the notion of uncertain data service, whose can be explained by possible worlds theory [Sadri, 1991]. These uncertain data services are defined by their inputs and sets of their probable outputs. It is this set of possible outputs returned by an invocation which can be considered as possible worlds, each of these world being Dependant and having a probability value. They defined two different kinds of invocation of uncertain data services, conventional with certain input and probabilistic where inputs are presented containing uncertain data instances.

These works propose several methods and models to process uncertainty in the context of Web (XML, services or semantics) but none of them address the uncertainty that can appear while referencing information through Web. This a very common problem, which is usually skipped, decided arbitrarily by providers. Our approach propose a relevant and adaptable approach to enhance Web based applications with uncertainty awareness.

5.3 Uncertain Web Resources

In this section, we define an uncertain resource as a Web resource whose representation cannot be decided. Therefore, as we see it, these resources have several possible representations, each with an associated probability.

5.3.1 Definition

The semantics of uncertain Web resource can be explained based on the possible worlds theory [Sadri, 1991]. The probabilistic representation of a resources can be interpreted as a set of possible world (PW_1 ,..., PW_n). In our approach, we decide to call this generated possible world **possible Webs**. Every possible Web has a probability $prob(PW_i)$ of existence, and inside this possible Webs, data is considered certain.

In order to define the notion of uncertain resource, we rely on several assumptions:

1. Due to the REST principles, several representation of one URI (i.e. one resource) cannot coexist, so the possible representations of a resource must be mutually exclusive.
2. Since we are dealing with resources, the uncertainty should only affect resource representations, therefore, inside a given possible representation, every piece of data is considered certain. By extension, if a resource property has several possible values, it should appear into separate representations.
3. For the sake of simplicity, in this approach we consider that each possible representation of a given resource is represented according to the same model.

We define an uncertain resource $\widetilde{R}$ as follows:

$$\widetilde{R} =< uri_r, \{< rep_i, P_i > | i \in [1, n], \sum_{i=1}^{n} \left(P_i\right) \leq 1\} >$$

Where rep_i is one of the possible representation for resource $\widetilde{R}$. Since possible representations are mutually exclusive, $P_i \in]0; 1]$. A sum of probabilities lower than 1 indicates that some of the representations of the resource exist but their actual content is unknown (or does not exist). The probabilities are not part of representations, they are meta-data provided by the server. In some cases, it allows to provide a sub parts of the possible representation, if the probability is to low or if the resource provider wants to hide some of the representations. A probability sum lower than 1 also provide the possibility to consider a quantified (as in predefined) part of unknown.

As an example, the two possible representations of our book resource scenario shown in Fig. 5.1a represents different possible Webs in which the representation is certain. In each of these Webs, resources are usable as a classical certain resources. Note that the interpretation of the probabilistic representation is completely independent from one resource to another. The associated model we defined is the following:

every resource is independent, but each URI identifies a unique resource, which can only have one representation.

In order to interpret our uncertain resources, we adapt the popular uncertain database model *Block-Independent Disjoint* (BID) [Dalvi et al., 2011]. Our model specifies, on the one hand, that every possible representations of a resource are disjoint, and on the other hand, that resource interpretations are independent from one to another.

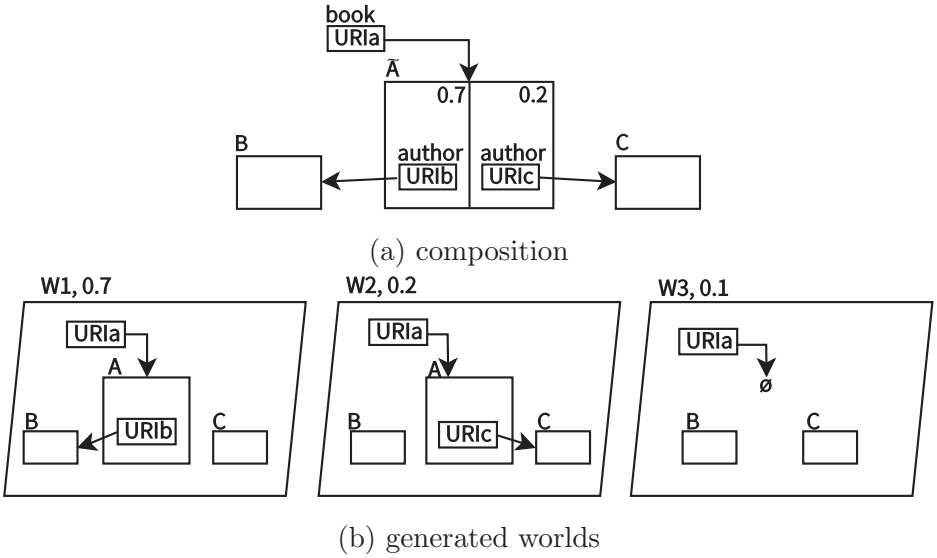


Figure 5.1 – A simple uncertain resource example

Fig. 5.1a shows how we interpret uncertain resources as a set of probable representations, each of these representation creating a possible Web in which it is the only representation of this resource. In this figure, we represent each resource a set of representations, the number in the upper right representing the probability of this representation. E.g. in the possible Web $W1$, resource A has a unique representation which contains a link to resource B; resource C still exists but is not connected to A.

Resources with non-provided representations

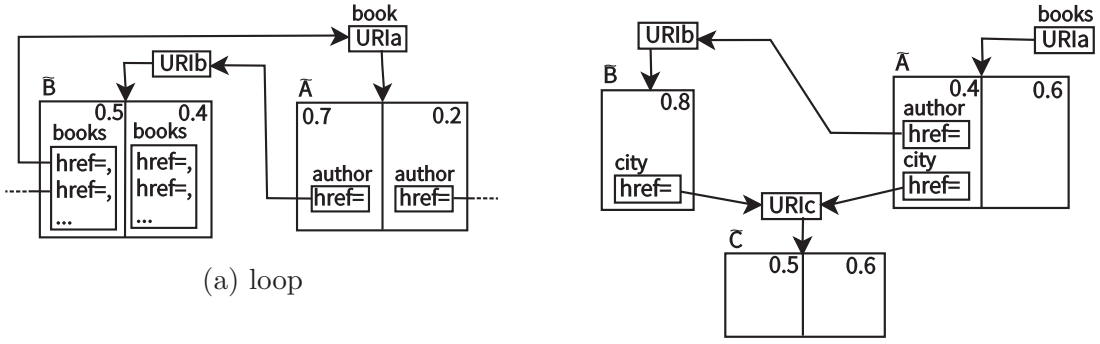
In possible Web $W3$, the uncertain resource $\widetilde{A}$ points to nothing, it is considered as a non-existing resource which does not have any representation, in our formalism, this unknown resource is noted $\emptyset$. As said by T. Fielding in his definition of REST [Fielding and Taylor, 2000] : "*A resource can map to the empty set, which allows references to be made to a concept before any realization of that concept exists*".

One resource can be unknown for several reason. The server that hosts the resource could had been shut down, the resource could have been removed from its location, the URI could be malformed.

Technically, a GET request over such an undefined resource leads to an HTTP error, as an example `404 not found` or `503 Service Unavailable` error status. However, when requesting the uncertain resource $\widetilde{A}$, we only obtain the known representation. In such cases, the sum of the provided resource representation probabilities does not make 100%. The unknown part will not directly affect our future algorithms, since we will only consider the known part of a resource, in this case, the sum of the probabilities for each query results will never complete a 100%. But this is something that often occurs, when manipulating probabilities.

5.3.2 Particular composition cases

Our previous example shows a quite simple example of scenario-based resource composition. The distributed state of resource-based applications, associated with our lightweight model, allow more complex compositions. Fig. 5.2 shows some examples of complex resource orchestration where uncertainty appear, and which our model could easily adapt to. These examples presents loop, redundancies and differences in models while navigating through hypertext. In these examples, heterogeneities can appear but are handled by our algorithm in the next section.



(a) loop

(b) dependencies

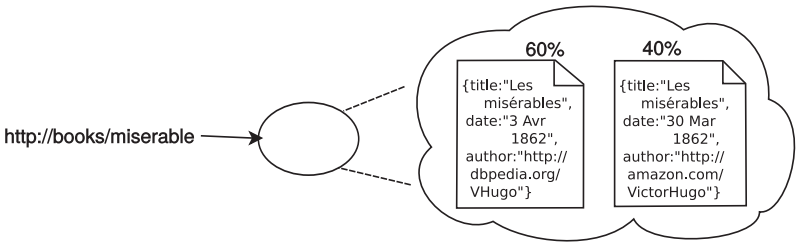
Figure 5.2 – Particular uncertain resource examples

Fig. 5.2a shows a situation where it may exists a loop in the request path. Our algorithm only dereferences resource once, protecting us from looping infinitely through hypertext path. In Fig. 5.2b, the resource composition present a duplicate resource.

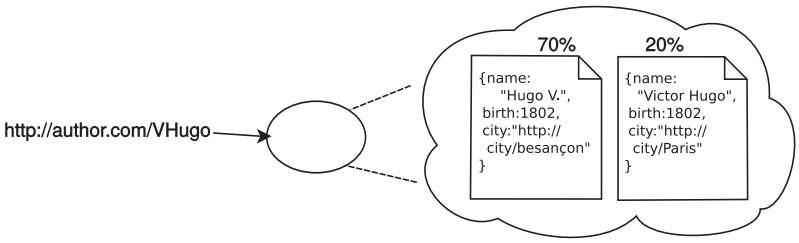
The important specificity here is related to this duplicate resource, and is handled by our model, which specifies that a resource only have one representation in a possible Web.

5.3.3 Programmatic representation of uncertain resources

In order to provide a way to handle uncertain resources, we proposed a formalism to physically represent them. We present a representation model, where we provide every possible representation of an uncertain resource. In this model, we give a probability to all these representations.



(a) Uncertain Book



(b) Uncertain Author

Figure 5.3 – Scenario based uncertain resources

Listing 5.1 and 5.2 shows the JSON representations of the uncertain resources presented in Fig. 5.3.

```
[ {p:0.6, r:{ title : "Les miserables",
              date: "3 Avr 1862",
              author: "http://dbpedia.org/VHugo"}},
  {p:0.4, r:{ title : "Les miserables",
              date: "30 Mar 1862",
              author: "http://dbpedia.org/VictorHugo"}} ]
```

Listing 5.1 – JSON representation of an uncertain book resource

```
[ {p:0.7, r:{ name: "Hugo V.",
              birth: 1802,
              city: "http://city/besancon"}},
```

```
{p:0.2, r:{ name: "Victor HUGO",
            birth: 1802,
            author: "http://city/paris"}} ]
```

Listing 5.2 – JSON representation of an uncertain author resource

5.3.4 HTTP Request over uncertain resources

In order to handle uncertain Web resources, there is a need for the client, who requests the uncertain resource to be able to understand this resource. We call *uncertain-aware* a client that is able to request and understand uncertain resource. In the same way, a client which does not know (or does not care about) what is an uncertain resource should be able to work with that resource. In order to respect the Web principles, and to provide a possibility to make client uncertain-aware, we rely on content negotiation to serve our uncertain resources.

Content negotiation is a mechanism provided by HTTP that allows to serve different *versions* of the same resource representation (i.e. at the same URI), to fit with the client (see *RFC7231*[11]). Through content-negotiation, a client is able to tell the server that it is able to compute uncertain resources. Doing so, we are able to enrich classical resources, and to manipulate both their certain and uncertain representations. We make a difference between classical and uncertain-aware GET requests. In order to differentiate standard GET requests and GET requests asking for uncertain-aware data, we propose the notation $\widetilde{GET}$, which describes a GET request from an uncertain-aware client (i.e. using specific headers to request uncertain representations).

Let $\widetilde{R}$ an uncertain resource deployed at uri_r, we defined the following expected behaviors:

$$GET(uri_r) :=< rep_r >$$

$$\widetilde{GET}(uri_r) := \{< rep_1, P_1 >, \ldots, < rep_n, P_n >\}$$

Listing 5.3 and 5.4 show examples of GET and $\widetilde{GET}$ over uncertain resources.

```
GET http://book.com/miserables HTTP/1.1
Accept: application/json

{ title : "Les miserables",
  author: "http://dbpedia.com/writer/Hugo"}
```

11. https://tools.ietf.org/html/rfc7231#section-3-4

Listing 5.3 – GET the certain (yet arbitrary) representation of our book

```
GET http://book.com/miserables HTTP/1.1
Accept: application/json
X−Accept−Uncertain: true

[ {p:0.4, r:{ title : "Les miserables",
              author: "http://dbpedia.com/writer/Hugo"}},
  {p:0.6, r:{ title : "Les miserables",
              author: "http://amazon.com/author/VictorHugo"}} ]
```

Listing 5.4 – GET the uncertain representation of our book

In our approach $\widetilde{GET}$ is **not** defining a new HTTP method. $\widetilde{GET}$ is only a shorter notation of a standard GET with specific headers. $\widetilde{GET}$ acts as a standard GET with a specific HTTP header which we define as $X-Accept-Uncertain : true$. We choose to define a specific header to avoid interference with the standardized usage of the *accept* header. Indeed, the *Accept* header is the classical header for content negotiation, as it is used to specify an expected mime-type for the resource representation. The good practice is then to specify an adhoc specific header to respect the HTTP standards (see *RFC7231*[12]).

NB: How providers define the certain representation of an uncertain resource is not a problem we address in the scope of this work. We only provide the possibility to do it.

Working with certain resources

In addition to this, using content negotiation will provide a possibility to work with certain resources, where the response provide only one representation with a probability of 1. Let R_c a certain resource at uri_c, we have the following:

$$GET(uri_C) :=< rep_c >$$

$$\widetilde{GET}(uri_c) := \{< rep_c, 1 >\}$$

Listing 5.5 and 5.6 show examples of certain and uncertain-aware manipulation of certain resources.

12. https://tools.ietf.org/html/rfc7231#section-5.3.2

```
GET http://book.com/treasureisland HTTP/1.1
Accept: application/json

{  title :  "Treasure Island",
   author: "http://dbpedia.com/writer/rl-stevenson"}
```

Listing 5.5 – GET a certain resource

```
GET http://book.com/miserables HTTP/1.1
Accept: application/json
X−Accept−Uncertain: true

[ {p:1,  r:{  title :  "Treasure Island",
              author: "http://dbpedia.com/writer/rl-stevenson"}} ]
```

Listing 5.6 – GET the uncertain representation of a certain resource

5.3.5 Composing uncertain Web resources

It is important to note that uncertain resources can link to other resources, which can also be uncertain resources. In such cases, we generate an uncertain composition between resources.

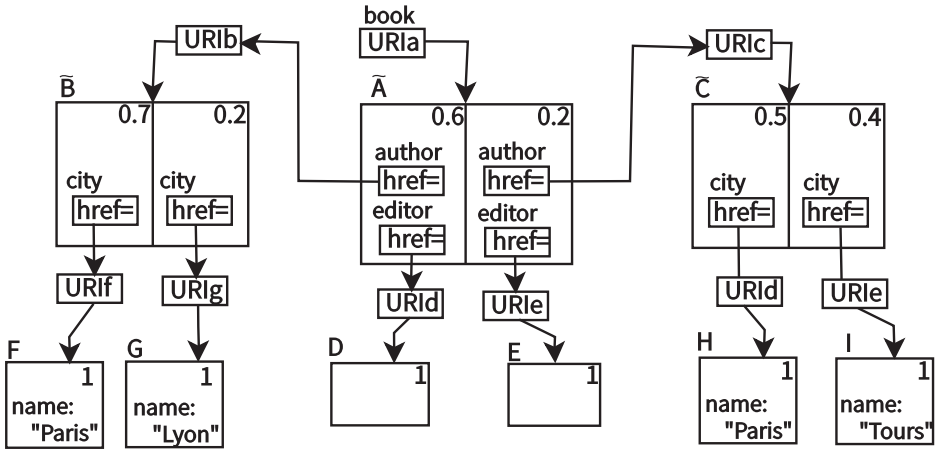
Let q a composition of Web resources Ri, with URI_i being R_i URI, and F_i^n the n^{th} possible representation of resource Ri. In this composition, each possible representation leads to the generation of a new possible Web. The probability of this possible Web is derived from the probabilities of its involved representations/ In this computation, we have to take into account, the resource representation involved. Since we consider that our resources are independent, we compute the resulting probability as a product. Let a possible Web W_x involving the set of representations $\{rep^1, ..., rep^n\}$, the probability of the resulting Web is computed as follows:

$$P(W_x) = \prod_{i \in [1,n]} \Big(prob(rep^i) \Big)$$

where $rep^i \in Card(W_x)$, the representations involved in W_x. The probability of the unknown representation of a resource R_a is computed as follows:

$$prob\big(rep_a^x\big) = 1 - \sum_{i=1}^{n} prob\big(rep_a^i\big)$$

where rep_a^i are the different representations of resource R_a. Fig. 5.4a shows a more complex example, where resource can link to different certain and non-certain



(a) composition

$B^1 \leftarrow A^1$ C F G D E H I $W_1, 0.42$	$B^X \leftarrow A^1$ C F G D E H I $W_3, 0.06$	B $A^2 \rightarrow C^X$ F G D E H I $W_6, 0.02$
$B^2 \leftarrow A^1$ C F G D E H I $W_2, 0.12$	B $A^2 \rightarrow C^1$ F G D E H I $W_4, 0.1$	B A^X C F G D E H I $W_7, 0.2$
	B $A^2 \rightarrow C^2$ F G D E H I $W_5, 0.08$	

(b) generated Webs

Figure 5.4 – Uncertain composition and world generation

resources. From this scenario, as shown in Fig. 5.4a, we generate all the possible Webs that are derived from the resources involved. In this figure, A^X, B^X and C^X represents the unknown part of each resource. WHat it means, technically, is this URIs points to nothing. Each probability is computed thanks to the formula in previous Subsection. As an example, the probability of possible Webs W_4 and W_6 are $prob(W_4) = prob(A^2) \times prob(C^1) \times prob(H) \times prob(E) = 0.2 \times 0.5 \times 1 \times 1 = 0.1$ and $prob(W_6) = prob(A^2) \times prob(C^X) \times prob(E) = 0.2 \times (1 - 0.5 - 0.4) \times 1 = 0.02$.

In this section, we introduced the concept of uncertain Web resources, presented a model and an algebra to compute the probability of uncertain resource composition. In the next section, we describe the evaluation of a query in such a context, we define how to interpret an uncertain query and how to compute and aggregate its results.

5.4 Query evaluation

In this section, we present our approach to aggregate data from uncertain resources thanks to hypertext navigation. We show how to interpret a data query in the context of uncertainty.

5.4.1 Interpreting query as resource paths

In the context of Web, searching for an information is commonly performed by navigating resources, following a natural path through these resources. Formally, we define a data query as an ordered set of resource requests, following this path. Resources URIs are dereferenced in order to get a representation, from which we extract new URIs, and so on, until we find query answer. Following our example scenario, to answer the query: *What is the date and city of birth of the writer of the book "Les Miserables" ?* The execution of this path in a classical RESTful composition is detailed in Fig. 5.5

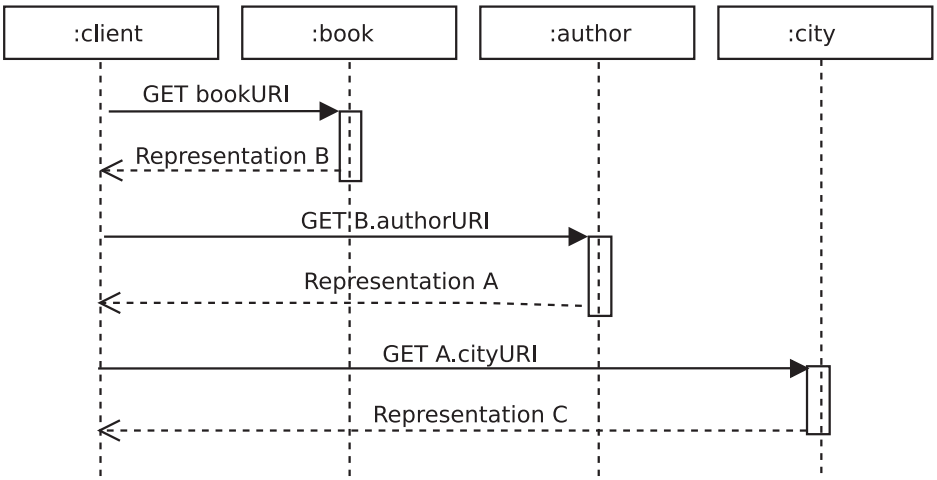


Figure 5.5 – Query answering in RESTful compositions

In order to fulfill this path following, we must assume that the representation we manipulate specify the necessary semantics and information about their content. For example, when retrieving a book resource and in order to reach its author, it is important that the `author` functional property of the object we retrieve is specified, and that this author is actually the writer of the book we visit. In our scenario, the representations we manipulate are represented in the JSON-LD format [Lanthaler and Gutl, 2012].

Answering a query in an uncertain context means following the same path, but

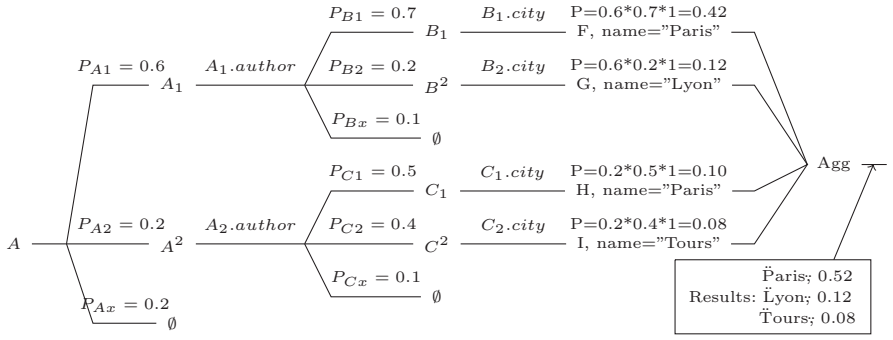


Figure 5.6 – Generating tree pattern while navigating between resources

through generated possible Web. Each Web provides a unique result, which is given with the probability of this possible Web. All results are aggregated to answer the query.

Generating each of these possible Web, i.e. combining and storing each combination of representation in memory in order to evaluate query path in each one is a time and memory consuming task. There is a need for an approach that allows to aggregate these results directly without having to generate the possible Webs.

5.4.2 Tree Pattern Path Evaluation

When dealing with uncertain resources, we follow our query path through the possible resource representations. This navigation through possible representation create a possibility tree pattern, where each branch is associated with a probability, i.e. the probability of the possible Web represented by this branch. Fig. 5.6 shows the tree pattern created from our book scenario.

There is a need for an algorithm to aggregate the result of this probability tree. What we propose here is an iterative algorithm to compute resulting probabilities from resource meta-data avoiding the possible Web generation. To do so, we create a GET_p operator to follow a stage-by-stage routing inside this tree.

GET_p takes as input a list of URIs from an n^{th} stage of the tree, and returns the possible resource representations from the $n + 1^{th}$ stage associated with probabilities taking n^{th} representation probability into account. Here is an example of GET_p usage:

$$GET_p\Big(\big[[A1.author, 0.6], [A2.author, 0.2]\big]\Big) = \tag{5.1}$$

$$\big[[B_1, 0.42], [B_2, 0.12], [C_1, 0.10], [C_2, 0.08]\big] \tag{5.2}$$

The GET_p operator executes the necessary sequence of HTTP requests over the given URIs, applies the probability formula and returns the set of representation-probability couples. Thanks to the mutually exclusive status of resource possible

Algorithm 1 GETp Algorithm

```
1: procedure GETp( input_uris: list of (URI,proba) couple )
2:     results ← List()
3:     for all (URI_i,prob_i) ∈ input_uris do
4:         GET(URI_i) returns uncertain resource R̃
5:         for all (representation,prob_r) ∈ R̃ do
6:             //Compute current probability
7:             prob_c ← prob_i * prob_r
8:             if representation ∉ results then
9:                 Add < representation, prob_c > to results
10:            else
11:                Update results[representation] += prob_c
       return results
```

representation, the method guarantees a safe composition, which means that result probabilities are coherent and the sums of this resulting probability are less or equal than 1 like verified in the following formula:

$$\sum_{i=1}^{n} \big(prob(result_i)\big) \in \big[0; 1\big]$$

where $< result_1, ..., result_n >$ is the resulting data set.

5.4.3 Final aggregation algorithm

Having introduced the uncertain operator GET_p, we are now able to finish our query resolution. The resolution algorithm process iteratively through the different stages of the probability tree. The evaluation algorithm will take a query and the URI of the first resource as input. The query will be converted into a path, i.e. a sequence of functional properties to extract from each resource. As an example, to get jump from book to authors, the functional property will be ":book hasAuthor:author". This path will help follow links from resource to resources.

The algorithm will process query path iteratively, using GET_p to dereference URIs of resource, and compute probabilities. In the case where a resource representation

Algorithm 2 Evaluation Algorithm

```
1: procedure EVALUATION( query, URI_0 )
2:     transform query in lists of properties (our path)
3:     // Make the first call (first URI is certain)
4:     result ← PROCESS_PATH( properties, < URI_0, 1 > )
5: procedure PROCESS_PATH( properties, input_uris )
6:     // properties is a list of functional properties (i.e. the path)
7:     // input_uris is a list of (URI,proba) couples
8:     rep ← GETp( input_uris )
9:     // Stopping condition, we reach the end of the path
10:    if prop[0] = ∅ then
11:        return rep
12:    else
13:        // Prepare new URI list
14:        new_uri_list ← []
15:        for all (representation, prob_r) ∈ rep do
16:            if representation has a property prop[0] which is an URI then
17:                // Get the property and add it to the new list
18:                new_uri_list[]←[representation.getprop(prop[0]),
   prob_r]
19:        return PROCESS_PATH( properties[:1], new_uri_list )
```

does not contain the desired content, this representation will be skipped. In the end, the resulting data set will contains the data that has been found, with the probability. Due to our probability tree representation of a query evaluation, the probability of not having a result will be taken into account, in this case, the probability will be lesser than 1.

In this section, we envision the complete execution process of a query evaluation under the existence of uncertainty. In the next section, we present some implementation detail.

5.5 Implementation

We propose an implementation for the GET_p algorithm and the computation algorithm. As introduced before, our approach relies on REST principles, so we are able to use any HTTP client to access our uncertain resources. What we propose here, is an implementation of the GET_p algorithm as well as an implementation of our evaluation algorithm, which will perform the necessary HTTP requests.

Here is an example of an HTTP request, using content negotiation, to an uncertain resource:

```
curl --header "X-Accept-Uncertain: true" "http://uri/resource"
```

Listing 5.7 – GET the uncertain representation of a certain resource

In order to keep our approach reusable, and to allow integration with other RESTful approaches, we implemented the GET_p and $COMPUTE$ algorithms as RESTful services. Service calls are made through POST, and GET retrieves a user-friendly description of the service. We propose a Web interface to execute simple SPARQL queries. Our prototype, resources and scenarios are publicly available for testing at the following URL: http://liris.cnrs.fr/~pdevetto/uncert/index.php.

We deployed and hosted our scenarios and proposed a Web interface to execute simple Sparql queries.

Here is an example of query, corresponding to our scenario:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?city_name WHERE {
    ?book a dc:book ;:hasAuthor ?author .
    ?author a dc:author ;: hasCity ?city .
    ?city a: city ;: hasName ?name .
}
```

Listing 5.8 – Query 1.1 involving the different concepts

Our *compute* algorithm implementation will transform this query in a list of concepts to extract from resource to resource,creating our path descending through the possibility tree. Our implementation uses the ARC2 SPARQL Parser to extract query concepts.

5.6 Evaluation

In order to evaluate our approach, we focus on the processing time of our algorithms. For this purpose, we hosted RESTful services serving uncertain Web resources in JSON-LD [Lanthaler and Gutl, 2012] over linked data dumps from the SWDF corpus (http://data.semanticweb.org), representing ESWC2015, ISWC2013, and WWW2012 conference semantic data (author, proceedings, etc.).

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX al: <http://liris.cnrs.fr/~pdevetto/uncert/>
PREFIX swrc: <http://swrc.ontoware.org/ontology#>
SELECT ?similararticle WHERE {
    ? article  a swrc:InProceedings ;
          dc:subject ?subject  .
    ?subject dc: inarticle  ? similararticle   .
}
```

Listing 5.9 – Query A: Articles that share the same subject than another article (by ID)

We created three different scenarios (use case workflows) involving a different amount of resources and with different graph complexities. Starting from a *proceedings article*, the first query shown in Listing 5.9 retrieves all the *articles* that share the same *keywords*. The second workflow (see Listing 5.10) retrieves all the *articles* written by at least one same *author*. Finally, the third workflow retrieves the *authors* that have written at least one *article* with one similar *keyword*.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX al: <http://liris.cnrs.fr/~pdevetto/uncert/>
PREFIX swrc: <http://swrc.ontoware.org/ontology#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
SELECT ?others WHERE {
? article  a swrc:InProceedings ;
   dc: authorlist  ? authorlist  .
? authorlist  a dc: authorlist  ;
   dc:author ?author .
?author a dc:author ;
    foaf : made ?others .
}
```

Listing 5.10 – Query B : articles with same keywords than given article

We executed all the workflows with 30 different proceeding articles as input data. Comparing uncertain workflow executions with the same workflow in a certain context has no meaning, because the number of HTTP request will grow exponentially. In our evaluation, we evaluate the ratio of network latency in the total execution cost of a workflow. We show that the processing cost of our solution is negligible compared to the network cost. The obtained results show the following: while workflows become more complex, the number of HTTP requests grows, and the processing time is more and more negligible, compared to HTTP latency. Under a global execution time of 2 seconds, processing time is less than 5%. After 3 seconds, it never exceeds 1%. On top of that, as long as input resources define coherent representations (and correct probabilities), no matter the query, it always generates a safe result set, with relevant values and probabilities.

5.7 Conclusion

There is a need to provide a solution for Web users to be able to handle data uncertainty while they navigate through hypertext. Taking the uncertainty into account will definitely improve the way, we process and trust the huge amount of information available on the Web. In this chapter, we address the need for a solution to handle data uncertainty while referencing and navigating resources on the Web. In this chapter, we propose a model to represent uncertain Web resources, considering uncertain resources as specific resources which could have several possible representations given a probability. We define these possible representations as mutually exclusive and rely on the possible world theory to interpret these representations as being part of a set of possible Webs.

On top of that, we rely on the uncertain Web resource model to propose an algebra for interpretation and evaluation of data queries in uncertain resource compositions. In this context, we define data queries as paths of Web resources, and propose an algorithm to avoid generating the possible Webs induced by the existence of possible resource representations. While respecting our probability model, we provide a computing algorithm, allowing to retrieve and compute the uncertainty associated with data query response.

Perspective of this approach includes opening our approach in order to deal with more complex scenarios, where possible representations could be actual Web resources with URIs. This way, we could construct a model based on hypertext navigation to define a resource according to a set of others, giving a possibility to represent the probable equivalence of resources. This approach will help adapt our architecture to handle the uncertainty that could appear on the Web, especially before and after data source combination.

Chapter 6

General conclusion

These last years have seen a growing interest in sharing and opening data to the World (to the Web). On top of that, numerous data sources are fed every day by users to spread knowledge and culture, or to organize interests and hobbies on platforms provided for this purpose. This new way of using the Web, associated with the growing number of APIs and endpoints making this data available, through machine readable formats, has led providers to try to add values to a huge amounts of freely available data. They collect, organize and reuse this data into mashups and services to provide useful tools and platforms. This upcoming possibilities have forced companies to adopt new data-driven strategies, adding more structure and semantics to draw benefits from data on the Web, combining them with their internal data. A lot of approaches and models have been proposed such as the semantic Web or the linked data initiative to connect data together, in order to help combine these data and extract highly valuable information, in response to a specific need.

Despite this advance and the availability of these sources, there is still a need for an approach which could automate some of the required process to provide techniques to handle the diversity and to help users to answer their questions. In this thesis, we focused on the variety of data source characteristics to propose an approach that helps automate the combination of heterogeneous multi-origin data sources. We proposed an adaptive resource-oriented approach (based on the usage of HTTP and Resource or Restful services) to automatically combine these data sources based on their logical and physical characteristics.

6.1 Summary of challenges

In order to build this approach, we identified the following scientific challenges. First of all, we focused on the need for adaptation provided by data source diversities and we identified the following challenges and scientific locks to address.

1. Dynamic and transparent data source management. It must be possible to transparently add or remove a data source at runtime without any need for hard coded information. There is a need for a formalism to represent this diversity of data sources.
2. Dynamic data processing. The solution needs to adapt at runtime to data sources that require different processings (large data volume, frequent update, latency).

Once these data-source challenges had been solved by introducing our first solution to automatically handle data access, we implemented the strategies we proposed in a software architecture, focusing on scalability and responsiveness. To do that, the solution had to be scalable and support a large number of data sources, and a variety of data source types, while offering low response time. This led to the following new challenges:

3. Given a sequence of tasks to execute, how to adapt the orchestration of the different tasks in order to optimize the global process in terms of response time and relevance.
4. How to implement such a mechanism in a software architecture, providing the flexibility and the adaptability that our solution requires in order to adapt to every kind of scenario.
5. It was necessary to propose a flexible solution that could easily adapt to future scenario needs.

Finally, when the data access and combination challenges mentionned above had been solved, we focused on data quality, by considering the uncertainty that could appear in Web context, and tried to solve the following questions:

6. What is the most relevant way to represent uncertainty on the Web ?
7. In what way could this affect Web browsing and automatic information retrieval (hypermedia navigation) ?

In the following, we summarize the contributions we proposed to answer these questions.

6.2 Contribution overview

In this dissertation, we addressed different challenges in order to propose an adaptive approach to automatically combine multi-origin data sources. Our approach allowed us to focus more on data source capabilities to improve the global integration process. These capabilities will affect the way we process the data, as implemented in our adaptive Web architecture. This report covered the different aspects of the solutions we propose to cover these issues. We summarize our major contributions below.

1. In order to focus our integration on data sources and to answer the dynamic and transparent data source management challenges, we first proposed a formalism to help the representation of the variety of data source characteristics. We defined a meta-model, to represent these characteristics, whether they be functional (URI, authentication, format, etc.) or non-functional (volume, latency). This meta-model separates our characteristics into two groups: characteristics of data sources, and characteristics of data itself, providing a solution to represent the necessary information to retrieve data, and later to process the data.
2. Focusing on the formalization of these characteristics, we defined several scenario-based models, to represent data sources and data. In our scenario, we presented a set of mandatory and optional characteristics as a set of key value attributes. With help from these models, we addressed the challenges associated with the need for adaptation, by defining some strategies that will help to adapt data source access (connection, extraction, ...) and data processing according to the specificity of each data source. These strategies include characteristic-based optimizations that implement different ways of extracting data.
3. Relying on a set of tasks that are required to perform a complete integration process and relying on our models and strategies, we proposed a software architecture under the form of a resource-oriented architecture, where each component is freely accessible through REST via its URI. Doing so, we provided the possibility of separating each step of the process into small components that can be distributed over a network. This flexible architecture helped us to adapt the process according to scenario contexts.
4. Relying on our flexible resource-oriented architecture, we assumed that task orchestration can be done in an optimized way, by automatically adapting to data sources and characteristics. We relied on a workflow adaptation technique,

where we processed a generic workflow that was adapted on-the-fly at runtime, based on the execution context. According to the characteristics of data sources, or metadata, the different tasks (i.e.: their execution) were prioritized amongst others (integration before semantic annotation, etc.), and removed or duplicated (duplicated filtering, before and after integration, etc.). This workflow adaptation allowed us to make the process faster, better, and more precise, depending on each case.

5. In order to prove the adaptability and flexibility of our approach, we overviewed our models and architecture, and proposed an example of architecture optimization to improve responsivity when manipulating large data volumes. In this contribution, we argued that large volumes can alter the complete process, especially in a Web based approach, where architecture components (i.e.: services and resources) are distributed over a network. In order to respond to this challenge, we proposed an optimization of our architecture based on the usage of distributed storage, to which the data was transfered only once. Services generated the process to execute upon the data, and this process was executed directly on data location. Doing so, only metadata was transfered through the architecture, which limited the data transfer through the architecture.

6. Having defined our architecture, we focused on a data quality issue. We relied on the fact that data uncertainty can appear when different data sources provide contradictory information about the same entity. In order to take this uncertainty into account, in our context, we proposed an approach to represent this uncertainty on the Web. To do so, we presented a model to create uncertain Web resources, as being resources which could have different and separated possible representations, associated with a probability of truth. We proposed an interpretation model to handle these uncertain resources, based on the fact that possible resource representations are mutually exclusive, and that resources must be independent. Based on this probabilistic model, we showed how uncertainty can affect hypermedia navigation.

7. Relying on the previously presented concept of uncertain resources, we focused on how it could affect our approach, especially when it comes to data retrieval, and hypermedia query answering. In order to solve this, we proposed a set of algorithms to perform hypermedia query evaluation (aggregation during hypermedia browsing) in the presence of uncertain Web resources. We also proposed an algorithm allowing the performance of the sequenced dereferencing of same-

concept probable representations, taking probability into account. This algorithm allowed us to aggregate possible representations of resources about the same concept, computing the relevant probabilities. Based on this algorithm, we proposed an iterative algorithm which is capable of browsing uncertain Web resources while aggregating data along the way and computing relevant probabilities, according to a query. We implemented these algorithms and deployed them as RESTful services. Our approach take a request as input, and transform this request into a semantic path amongst several concepts, and one or several entry points as resource URIs, that could be uncertain or not.

Following these contributions, our future work will include additional evaluation over larger data sets and exploring issues related to data management such as freshness issues. Our approach could also be improved with reasoning for a better recognition of inconsistent or imprecise data.

6.3 Research perspectives

Today, there are many approaches proposed in very specific domains, and we could gain benefit from crossing these domains to propose hybrid solutions. In this work, we propose a generic data source based solution which is at the crossroads of many different application fields. This approach relies on advances in **semantics and linked data** technologies, to propose a **resource-oriented architecture** that handles **data source integration**, focusing on the **uncertainty** which could appear while processing data. In the following paragraphs, we identify several directives for future work in this field.

One of the major interesting perspectives of this work is to improve the semantic management of data in our approach. This approach relies on existing techniques to provide semantics through the annotation or transformation of existing data. However, this requires a human input to provide the semantic bridge through matching or semantic description. It could be relevant to propose an approach to help the recognition of semantics and characteristics for new data sources. Coupling our approach with a reasoner, entity recognition or natural language processing techniques, it could be possible to propose schema and semantics recognition. In the end, improving this data source recognition could allow us to perform data source discovery over the Web. This data source discovery approach could feed an index and provide either a data source search engine, or an automated data source composition in response to a specific query,

which would be very useful to the everyday user of the Web.

On top of that, we clearly see that hypermedia driven applications are one of the most exciting topics in the Web community today [Lanthaler and Gütl, 2013]. This subject, which somehow coincides with the perspectives we presented in the last paragraph, consists of applications that are decoupled in such a way that the client automatically chooses the next resource to request, according to its description and to the context of the current execution. In our context of *question answering* by combining data sources, our solution could be adapted in such a way so that the client is driven from resource to resource, gathering the necessary information to answer his query, or to fulfill his objective. It could be interesting to improve this kind of Web discovery from resource to resource, aggregating data along the way. In this perspective, our client could transport a kind of inventory of harvested data. This way, the final user, would only have to extract an organized, structured and semantically annotated summary of the research session he just finished. By storing this hypermedia browsing result set as a resource, it could be shared with users, or saved on the Web for future purposes.

Finally, whereas data uncertainty has been studied a lot in the context of databases and services, our contribution in the context of the Web raises a lot of new questions about data uncertainties on the Web. Our approach proposes a definition of uncertain web resources which allow a basic interpretation of uncertainty in a Web context. It also raises a lot of new challenges for further research. First of all, how can it be possible to automatically retrieve the possible representation of a resource according to a subject? It could be interesting to propose a discovery approach that could be able to automatically aggregate an uncertain resource about a given subject, and compute a probability score for each of the retrieved possibilities. Secondly, since our approach proposes representing the different resource representations according to the same model, it could be interesting to extend this concept to a wider area of the Web. It could even be possible to define a representation as an ordered and weighted set of existing resources of the Web. Doing so, we could extend the concept of uncertain resources, by enhancing its definition in a Linked Data way, thus providing the user with relevant links to follow in order to retrieve additional information about a resource.

Publications list

Publications related to this thesis (chronological)

1. P. De Vettor, M. Mrissa & D. Benslimane : Modeling and Composing Uncertain Web Resources. In The Semantic Web - ESWC 2016 Satellite Events - Revised Selected Papers: 327-341.
2. P. De Vettor, M. Mrissa & D. Benslimane : Towards Definition and Composition of Uncertain RESTful Resources. In Proceedings of the Third Workshop on Services and Applications over Linked APIs and Data, SALAD @ ESWC 2016.
3. P. De Vettor, M. Mrissa & D. Benslimane: A Resource-Oriented Architecture to Handle Data Volume Diversity. In Proceedings of the CAiSE 2015 Forum at the 27th International Conference on Advanced Information Systems Engineering. CAiSE Forum 2015: 161-168
4. P. De Vettor, M. Mrissa & D. Benslimane: Models and Adaptive Architecture for Smart Data Management. In 24th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises. WETICE 2015: 164-169
5. P. De Vettor, M. Mrissa & D. Benslimane: Models and Architecture for Smart Data Management. Rapport de recherche RR-LIRIS-2014-017, http://liris.cnrs.fr/Documents/Liris-7004.pdf
6. P. De Vettor, M. Mrissa, D. Benslimane & S. Berbar: A Service Oriented Architecture for Linked Data Integration. In 8th IEEE International Symposium on Service-Oriented System Engineering. SOSE 2014: 198-203

Bibliography

[Adams et al., 2006] Adams, M., ter Hofstede, A., Edmond, D., and van der Aalst, W. (2006). Worklets: A service-oriented implementation of dynamic flexibility in workflows. In Meersman, R. and Tari, Z., editors, *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4275 of *Lecture Notes in Computer Science*, pages 291–308. Springer Berlin Heidelberg.

[Agrawal et al., 2010] Agrawal, P., Sarma, A. D., Ullman, J., and Widom, J. (2010). Foundations of uncertain-data integration. *Proc. VLDB Endow.*, 3(1-2):1080–1090.

[Alani et al., 2007] Alani, H., Dupplaw, D., Sheridan, J., O'Hara, K., Darlington, J., Shadbolt, N., and Tullo, C. (2007). Unlocking the potential of public sector information with semantic web technology. Event Dates: November.

[Amdouni et al., 2014] Amdouni, S., Barhamgi, M., Benslimane, D., and Faiz, R. (2014). Handling uncertainty in data services composition. In *Services Computing (SCC), 2014 IEEE International Conference on*, pages 653–660.

[Apache, 2013] Apache (2013). Apache metamodel : A data access framework. http://metamodel.incubator.apache.org/.

[AtosWorldline, 2013] AtosWorldline (2013). Smartdata.io : A dedicated solution to the data management. http://api.docs.v2.smartdata.io/.

[Auer et al., 2009] Auer, S., Dietzold, S., Lehmann, J., Hellmann, S., and Aumueller, D. (2009). Triplify: Light-weight linked data publication from relational databases. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 621–630, New York, NY, USA. ACM.

[Ba et al., 2014] Ba, M., Montenez, S., Tang, R., and Abdessalem, T. (2014). Integration of web sources under uncertainty and dependencies using probabilistic xml. In Han, W.-S., Lee, M. L., Muliantara, A., Sanjaya, N. A., Thalheim, B., and Zhou, S., editors, *Database Systems for Advanced Applications*, Lecture Notes in Computer Science, pages 360–375. Springer Berlin Heidelberg.

[Benslimane et al., 2008] Benslimane, D., Dustdar, S., and Sheth, A. P. (2008). Services mashups: The new generation of web applications. *IEEE Internet Computing*, 12(5):13–15.

[Berners-Lee, 2005] Berners-Lee, T. (2005). Notation3: Logic and rules on rdf. Technical report, Word Wide Web Consortium.

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 284(5):34–43.

[Bizer, 2004] Bizer, C. (2004). D2rq - treating non-rdf databases as virtual rdf graphs. In *In Proceedings of the 3rd International Semantic Web Conference (ISWC2004.*

[Bizer et al., 2007] Bizer, C., Cyganiak, R., and Heath, T. (2007). How to publish linked data on the web. Web page. Revised 2008. Accessed 22/02/2010.

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

[Brickley and Miller, 2007] Brickley, D. and Miller, L. (2007). The friend of a friend (foaf) vocabulary specification. http://xmlns.com/foaf/spec/.

[Brzeziński et al., 2011] Brzeziński, J., Danilecki, A., Flotyński, J., Kobusińska, A., and Stroiński, A. (2011). Workflow engine supporting restful web services. In Nguyen, N., Kim, C.-G., and Janiak, A., editors, *Intelligent Information and Database Systems*, volume 6591 of *Lecture Notes in Computer Science*, pages 377–385. Springer Berlin Heidelberg.

[Chappell, 2004] Chappell, D. A. (2004). *Enterprise service bus - theory in practice.* O'Reilly.

[Cheng et al., 2012] Cheng, R., Gong, J., Cheung, D., and Cheng, J. (2012). Evaluating probabilistic queries over uncertain matching. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 1096–1107.

[Cramer et al., 2004] Cramer, C., Schafferhans, A., and Fuhrmann, T. (2004). Peer-to-peer overlays and data integration in a life science grid. In *Proceedings of the First International Workshop of the EU Network of Excellence DELOS on Digital Library Architectures*, pages 127–138, Cagliari, Italy.

[Dalvi et al., 2011] Dalvi, N., Re, C., and Suciu, D. (2011). Queries and materialized views on probabilistic databases. *Journal of Computer and System Sciences*, 77(3):473 – 490. Database Theory.

[Das Sarma et al., 2008] Das Sarma, A., Dong, X., and Halevy, A. (2008). Bootstrapping pay-as-you-go data integration systems. In *Proceedings of the 2008 ACM*

SIGMOD International Conference on Management of Data, SIGMOD '08, pages 861–874, New York, NY, USA. ACM.

[Delen and Demirkan, 2013] Delen, D. and Demirkan, H. (2013). Data, information and analytics as services. *Decision Support Systems*, 55(1):359 – 363.

[Devresse and Furano, 2014] Devresse, A. and Furano, F. (2014). Efficient HTTP based I/O on very large datasets for high performance computing with the libdavix library. *CoRR*, abs/1410.4168.

[Dong et al., 2007] Dong, X., Halevy, A. Y., and Yu, C. (2007). Data integration with uncertainty. In *Proceedings of the 33rd international conference on Very large data bases*, pages 687–698. VLDB Endowment.

[Dorn et al., 2009] Dorn, C., Schall, D., and Dustdar, S. (2009). Context-aware adaptive service mashups. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pages 301–306.

[Dustdar et al., 2012] Dustdar, S., Pichler, R., Savenkov, V., and Truong, H. L. (2012). Quality-aware service-oriented data integration: requirements, state of the art and open challenges. *SIGMOD Record*, 41(1):11–19.

[Eckert et al., 2014] Eckert, K., Ritze, D., Baierer, K., and Bizer, C. (2014). Restful open workflows for data provenance and reuse. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, pages 259–260, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Erl, 2005] Erl, T. (2005). *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[Erl, 2009] Erl, T. (2009). *SOA Design Patterns*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

[Fagin et al., 2011] Fagin, R., Kimelfeld, B., and Kolaitis, P. G. (2011). Probabilistic data exchange. *Journal of the ACM (JACM)*, 58(4):15.

[Fielding and Taylor, 2000] Fielding, R. T. and Taylor, R. N. (2000). Principled design of the modern web architecture. In *Proceedings of the 22Nd International Conference on Software Engineering*, ICSE '00, pages 407–416, New York, NY, USA. ACM.

[Ford et al., 2011] Ford, A., Raiciu, C., Handley, M., Barre, S., and Iyengar, J. (2011). Architectural guidelines for multipath tcp development. In *IETF, RFC 6182*. Citeseer.

[Furth and Baumeister, 2013] Furth, S. and Baumeister, J. (2013). Towards the semantification of technical documents. In *FGIR'13: Proceedings of German Workshop of Information Retrieval (at LWA'2013)*.

[Grossman et al., 2005] Grossman, R. L., Gu, Y., Hong, X., Antony, A., Blom, J., Dijkstra, F., and de Laat, C. (2005). Teraflows over gigabit {WANs} with {UDT}. *Future Generation Computer Systems*, 21(4):501 – 513. High-Speed Networks and Services for Data-Intensive Grids: the DataTAG Project.

[Groth et al., 2014] Groth, P., Loizou, A., Gray, A. J., Goble, C., Harland, L., and Pettifer, S. (2014). Api-centric linked data integration: The open {PHACTS} discovery platform case study. *Web Semantics: Science, Services and Agents on the World Wide Web*, 29:12 – 18. Life Science and e-Science.

[Halevy et al., 2006] Halevy, A., Rajaraman, A., and Ordille, J. (2006). Data integration: The teenage years. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*, VLDB '06, pages 9–16. VLDB Endowment.

[Han et al., 2008] Han, L., Finin, T., Parr, C., Sachs, J., and Joshi, A. (2008). Rdf123: From spreadsheets to rdf. In *The Semantic Web - ISWC 2008*, volume 5318 of *Lecture Notes in Computer Science*, pages 451–466. Springer Berlin Heidelberg.

[Heath and Bizer, 2011] Heath, T. and Bizer, C. (2011). *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers.

[Herrmann et al., 2000] Herrmann, T., Hoffmann, M., Loser, K.-U., and Moysich, K. (2000). Semistructured models are surprisingly useful for user-centered design. In *Designing Cooperative Systems. Proceedings of Coop 2000*, pages 159–174.

[Hohpe and Woolf, 2003] Hohpe, G. and Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Hurst, 2001] Hurst, M. (2001). Layout and language: Challenges for table understanding on the web. In *Proceedings of the International Workshop on Web Document Analysis*, pages 27–30.

[Kimelfeld and Senellart, 2013] Kimelfeld, B. and Senellart, P. (2013). Probabilistic xml: Models and complexity. In Ma, Z. and Yan, L., editors, *Advances in Probabilistic Databases for Uncertain Information Management*, volume 304 of *Studies in Fuzziness and Soft Computing*, pages 39–66. Springer Berlin Heidelberg.

[Ko et al., 2012] Ko, R., Kirchberg, M., Lee, B.-S., and Chew, E. (2012). Overcoming large data transfer bottlenecks in restful service orchestrations. In *Web Services (ICWS), 2012 IEEE 19th International Conference on*, pages 654–656.

[Lanthaler and Gutl, 2012] Lanthaler, M. and Gutl, C. (2012). On using json-ld to create evolvable restful services. In *Proceedings of the Third International Workshop on RESTful Design*, WS-REST '12, pages 25–32, New York, NY, USA. ACM.

[Lanthaler and Gütl, 2013] Lanthaler, M. and Gütl, C. (2013). Hydra: A vocabulary for hypermedia-driven web apis. *LDOW*, 996.

[Leiba, 2012] Leiba, B. (2012). Oauth web authorization protocol. *IEEE Internet Computing*, 16(1):74–77.

[Lenzerini, 2002] Lenzerini, M. (2002). Data integration: A theoretical perspective. In Popa, L., Abiteboul, S., and Kolaitis, P. G., editors, *PODS*, pages 233–246. ACM.

[Levy et al., 1996] Levy, A. Y., Rajaraman, A., and Ordille, J. J. (1996). Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pages 251–262, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Lopez and Kotoulas, 2012] Lopez, V. and Kotoulas, S. (2012). Queriocity: A linked data platform for urban information management. In *International Semantic Web Conference (2)*, volume 7650 of *Lecture Notes in Computer Science*, pages 148–163. Springer.

[Maleshkova et al., 2010] Maleshkova, M., Pedrinaci, C., and Domingue, J. (2010). Investigating web apis on the world wide web. In *Web Services (ECOWS), 2010 IEEE 8th European Conference on*, pages 107–114.

[Minor et al., 2007] Minor, M., Schmalen, D., Koldehoff, A., and Bergmann, R. (2007). Structural adaptation of workflows supported by a suspension mechanism stand by case-based reasoning. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2007. WETICE 2007. 16th IEEE International Workshops on*, pages 370–375. IEEE.

[Mrissa et al., 2013] Mrissa, M., Sellami, M., Vettor, P. D., Benslimane, D., and Defude, B. (2013). A decentralized mediation-as-a-service architecture for service composition. *2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 0:80–85.

[O'Donovan et al., 2015] O'Donovan, P., Leahy, K., Bruton, K., and O'Sullivan, D. T. J. (2015). Big data in manufacturing: a systematic mapping study. *Journal of Big Data*, 2(1):1–22.

[Pivert and Prade, 2014] Pivert, O. and Prade, H. (2014). Querying uncertain multiple sources. In Straccia, U. and Cali, A., editors, *Scalable Uncertainty Management*, volume 8720 of *Lecture Notes in Computer Science*, pages 286–291. Springer International Publishing.

[Pivk, 2005] Pivk, A. (2005). Automatic ontology generation from web tabular structures. *AI Communications*, 19:2006.

[Pontieri et al., 2003] Pontieri, L., Ursino, D., and Zumpano, E. (2003). An approach for the extensional integration of data sources with heterogeneous representation formats. *Data & Knowledge Engineering*, 45(3):291 – 331.

[Richardson and Ruby, 2007] Richardson, L. and Ruby, S. (2007). *Restful Web Services*. O'Reilly, first edition.

[Rosaci et al., 2004] Rosaci, D., Terracina, G., and Ursino, D. (2004). A framework for abstracting data sources having heterogeneous representation formats. *Data & Knowledge Engineering*, 48(1):1 – 38.

[Rosenberg et al., 2008] Rosenberg, F., Curbera, F., Duftler, M. J., and Khalaf, R. (2008). Composing restful services and collaborative workflows: A lightweight approach. *IEEE Internet Computing*, 12(5):24–31.

[Sadiq et al., 2001] Sadiq, S., Sadiq, W., and Orlowska, M. (2001). Pockets of flexibility in workflow specification. In S.Kunii, H., Jajodia, S., and SÃ¸lvberg, A., editors, *Conceptual Modeling at ER 2001*, volume 2224 of *Lecture Notes in Computer Science*, pages 513–526. Springer Berlin Heidelberg.

[Sadri, 1991] Sadri, F. (1991). Modeling uncertainty in databases. In *Data Engineering, 1991. Proceedings. Seventh International Conference on*, pages 122–131.

[Sheth, 2014] Sheth, A. (2014). Transforming big data into smart data: Deriving value via harnessing volume, variety, and velocity using semantic techniques and technologies. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 2–2.

[Stankovski et al., 2010] Stankovski, V., Missier, P., Goble, C., and Taylor, I. (2010). Open workflow infrastructure: A research agenda. In *Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science*, Wands '10, pages 6:1–6:6, New York, NY, USA. ACM.

[Sun et al., 2014] Sun, Z., Strang, K. D., and Yearwood, J. (2014). Analytics service oriented architecture for enterprise information systems. In *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '14, pages 508–516, New York, NY, USA. ACM.

[Truong et al., 2011] Truong, H.-L., Dustdar, S., Goetze, J., Fleuren, T., Mueller, P., Tbahriti, S.-E., Mrissa, M., and Ghedira, C. (2011). Exchanging data agreements in the daas model. In *The 2011 IEEE Asia-Pacific Services Computing Conference*, pages 153–160. IEEE.

[van der Aalst and ter Hofstede, 2005] van der Aalst, W. and ter Hofstede, A. (2005). Yawl: yet another workflow language. *Information Systems*, 30(4):245 – 275. YAWL.

[van der Aalst et al., 2003] van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., and Barros, A. (2003). Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51.

[van der Pol et al., 2012] van der Pol, R., Boele, S., Dijkstra, F., Barczyk, A., van Malenstein, G., Chen, J. H., and Mambretti, J. (2012). Multipathing with mptcp and openflow. *High Performance Computing, Networking Storage and Analysis, SC Companion:*, 0:1617–1624.

[Venetis et al., 2011] Venetis, P., Halevy, A. Y., Madhavan, J., Pasca, M., Shen, W., 0003, F. W., Miao, G., and Wu, C. (2011). Recovering semantics of tables on the web. *PVLDB*, 4(9):528–538.

[Verborgh et al., 2016] Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G., and Colpaert, P. (2016). Triple Pattern Fragments: a low-cost knowledge graph interface for the Web. *Journal of Web Semantics*, 37–38:184–206.

[Weibel and Koch, 2000] Weibel, S. and Koch, T. (2000). The dublin core metadata initiative : Mission, current activities, and future directions. *D-Lib Magazine*, 6(12).

[Zheng et al., 2013] Zheng, Z., Zhu, J., and Lyu, M. (2013). Service-generated big data and big data-as-a-service: An overview. In *Big Data (BigData Congress), 2013 IEEE International Congress on*, pages 403–410.

Résumé long

La réutilisation des données semble aujourd'hui préoccuper la communauté scientifique, particulièrement dans le domaine du Web. De la façon dont nous le voyons, la réutilisation des données est l'une des solutions qui existe pour tirer parti des données brutes hétérogènes stoquées sous divers formats un peu partout sur le Web ou à l'interieur des systèmes d'information des sociétés.

Cette thèse porte sur l'intégration de données brutes provenant de sources hétérogènes sur le Web. L'objectif global est de fournir une architecture générique et modulable capable de combiner, de façon sémantique et intelligente, ces données hétérogenes dans le but de les rendre réutilisables. Ce travail est motivé par un scenario réel de l'entreprise *Audience Labs* permettant une mise à l'échelle de cette architecture.

Dans ce rapport, nous proposons de nouveaux modèles et techniques permettant d'adapter le processus de combinaison et d'intégration à la diversité des sources de données impliquées. Les problématiques sont :

— une gestion transparente et dynamique des sources de données
— un meilleur passage à l'échelle et de meilleurs temps de réponse par rapport au nombre de sources
— adaptativité aux caractéristiques des sources
— finalement, consistance des données produites (données cohérentes, sans erreurs ni doublons).

Pour répondre à ces problématiques, nous proposons différentes contributions:

— un méta-modèle (et des modèles) pour représenter et accéder aux sources de données
— une architecture orientée resource, mettant en oeuvre des workflow adaptatifs, en fonction des sources
— une approche permettant de prendre en compte l'incertitude dans un contexte Web

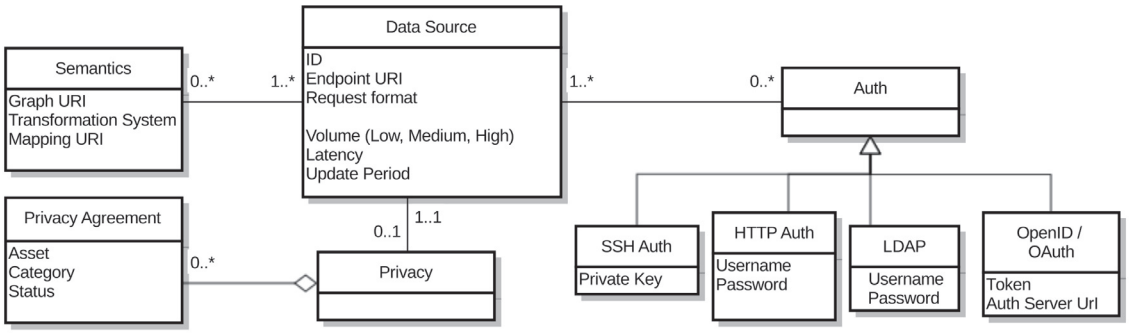


Figure 1 – Modèle de source de donnée basé sur notre scénario

Modèles et stratégies de traitements

Dans le but de rendre le système adaptable à la variété des sources de données, nous construisons un méta-modèle, permettant de construire des modèles de source de donnée.

Ce méta-modèle définit les éléments nécessaires à la confection de modèles pour representer les sources de données au travers d'un ensemble de caractéristiques. Ces caractéristiques peuvent être liées à l'accès aux données (URI, authentification, ...), aux caractéristiques structurelles des sources (sémantique, format de requête, schéma, ...), et enfin aux caractéristiques physiques des sources (latence, volume, ...). De plus, ce méta-modèle nous permet également de définir des modèles pour représenter les données elles-mêmes et notamment le contexte qu'elles transportent.

En se basant sur ces modèles, nous définissons les comportements spécifiques à adopter pour accéder aux données. Par exemple, sur des données à faible fréquence de mise à jour, mais sur à haute latence, il est nécessaire de mettre en place des systèmes de cache. D'un autre façon, les données à haute fréquence de mise à jour nécéssitent de conserver une durée de validité pour les données extraites.

Architecture orientée service

En se basant sur ces modèles et stratégies, nous proposons une approche de workflow adaptatif. Il s'agit donc de workflows dont l'ordre d'éxecution varie en fonction du contexte, c'est à dire en fonction des caractéristiques des sources de données. Le workflow adaptatif sera dćoupé en sous-parties pouvant être dupliquées, supprimées ou déplacées.

Nous proposons ensuite une architecture orientée resource, où les différents ser-

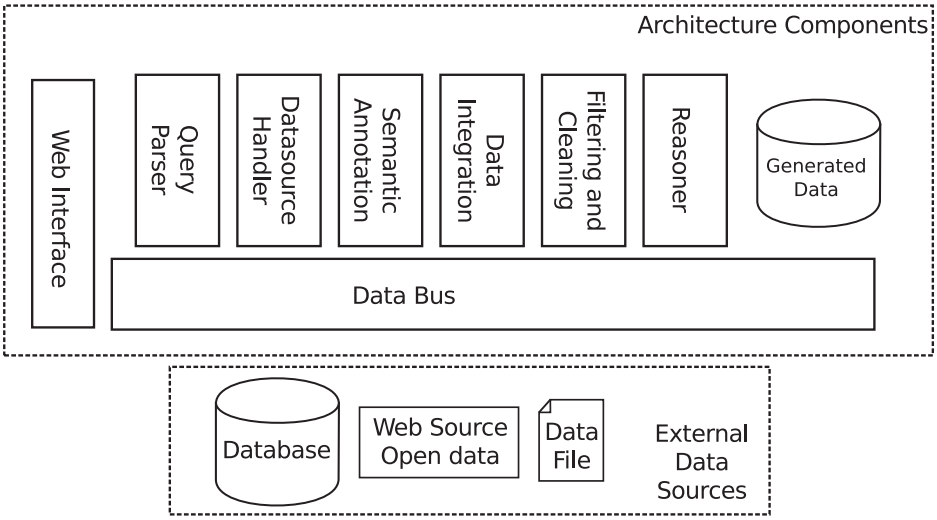


Figure 2 – Architecture orientée resources

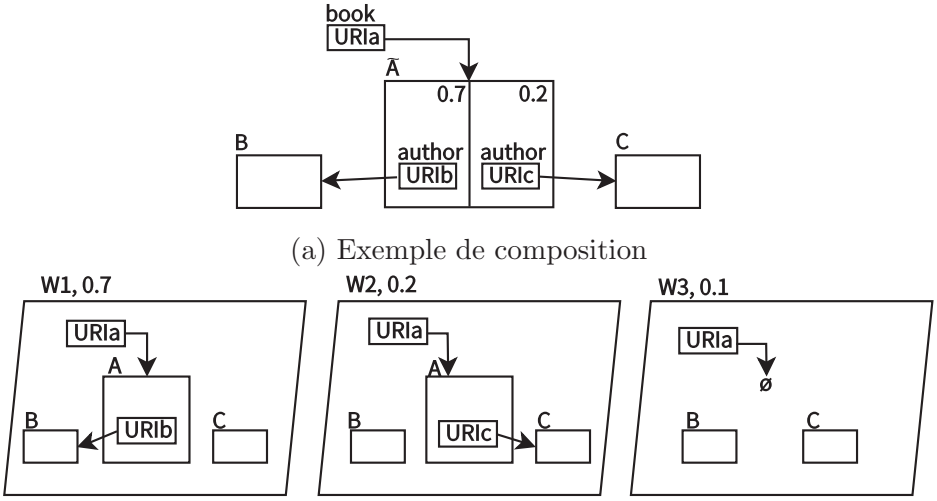
vices nécessaires au fonctionnement d'une processus global d'intégration (gestionnaire de sources de données, service d'annotation sémantique, service de combinaison des données) sont définis comme des services RESTful, accessibles par HTTP via leurs URIs.

En se basant sur les caractéristiques des sources, notre architecture adaptera le workflow d'intégration, orchestrant les différentes tâches du processus d'intégration de façon optimale en priorisant chacune des tâches. Ainsi, les temps de traitement et le volume des données échangées sont diminués.

Nous démontrons par la suite le coté adaptif de notre architecture, en proposant une optimisation permettant de modifier le comportement de l'architecture face aux gros volumes de données. Cette approche repose sur l'utilisation d'un stockage de données distribué, et sur le principe que chacun des composants envoye les traitements à éxécuter directement au niveau du stockage. De cette façon, seules les métadonnées transfèrent dans l'architecture, et les temps d'éxécution en sont réduits.

Resource Web Incertaines

Afin d'améliorer la qualité des données produites par notre approche, l'accent est mis sur l'incertitude qui peut apparaitre dans les données sur le Web. Nous proposons un modèle probabilistique, permettant de représenter cette incertitude, à travers le concept de resource Web incertaine, basé sur un modèle probabiliste ou chaque resource peut avoir plusieurs représentation possibles, avec une certaine probabilité.



(a) Exemple de composition

(b) Mondes générés

Figure 3 – Exemple de ressource incertaine simple

Chacune des ces représentation possible génère un Web possible dans lequel les données seront considerées comme certaines. L'éxecution de requêtes (i.e., la navigation hypermédia) dans ces Web certains devient alors complexe. Afin de résoudre ces requêtes, nous proposons de représenter ces liens hypermédias sous la forme d'arbres. De cette façon, il est possible de définir un algorithme itératif afin d'éviter la génération des Web possibles. Une étape d'aggrégation intermédiaire nous permet également d'optimiser l'éxécution de nos requêtes.

Cette approche sera à l'origine d'une nouvelle optimisation de l'architecture pour permettre de prendre en compte l'incertitude pendant la combinaison des données.

Conclusion

Nous proposons différentes implémentations des ces approches, qui nous permettent d'étudier les temps de réponse, et ainsi d'évaluer nos approches. Nous avons adressé nos challenges de la façon suivante.

Nous avons tout d'abord proposé un méta-modèle pour représenter la variété des sources de données, que ce soit au niveau de l'accès/extraction, de la structure des données ou des contraintes physiques. Cette formalisation nous permet d'adapter les traitements (stratégies) à la variété des sources.

L'orchestration des différentes tâches du processus d'intégration étant faite de manière optimisée, nous proposons une solution pouvant être mise à l'échelle, et présen-

tant des temps de réponse raisonables malgré le nombre de sources de donnée.

Finalement, notre approche de resource Web incertaine, nous permet de nous intéresser aux problématiques de qualité des données en proposant une solution pour représenter l'incertitude qui peut exister sur le Web. Cette approche sera à la base d'une nouvelle optimisation de l'architecture, afin de prendre en compte l'incertitude lors de la combinaison des données.