University of Milan-Bicocca
Department of Information, Systems and Communications
Doctor of Philosophy in Computer Science
Cycle: XXVII
Coordinator: Prof. Stefania Bandini

User-Driven Composition of Web APIs

Bridging the Gap between Users' Requirements and Technology Constraints

By

Meherun Nesa Lucky
Student ID: 761118

Supervisor: Prof. Flavio De Paoli
Tutor: Prof. Stefania Bandini

A dissertation submitted for the degree of *Doctor of Philosophy*

Academic Year 2016/2017

Abstract

In this dissertation, we present a novel approach for user-driven composition of Web APIs by adopting end-user development paradigm. Automated Web service or Web API composition driven by end-users requires dealing with three major research thrusts: (i) heterogeneity of the Web services due to different protocols they use, (ii) automated addition of semantic annotations in the service descriptions and (iii) involvement of all groups of end-users in the composition process by providing an easy and accessible environment according to their requirements. Nowadays, variety of services on the Web is increasing considering pervasive computing environments where different devices often use different technologies, which make the communication complex and sometimes even troublesome. Sensor Web has a central role in addressing new sources of information that need to be dynamically integrated into systems. Although, the HTTP protocol has been used as a universal mean for tunneling messages in business-to-business scenarios, interoperability between protocols such as WSDL/SOAP for Web Services or OGC/SOS/SPS for sensors is missing. We consider this issue and investigate the correct and complete use of the HTTP protocol to publish, manage, and operate services on the Web by fully exploiting the REST (REpresentational State Transfer) principles. We first propose an REST-based framework for providing a common way of interactions among heterogeneous Web Services. To accomplish this, we employ a common RESTful interface for services interactions by defining URI scheme to describe resources, the HTTP idioms (methods, headers, status codes) to interact with the service, the domain application protocol, and the media type for hypermedia-driven interactions. Although there are several description formats and tools available to describe REST Web APIs in human and machine readable formats, adding semantic annotation to these descriptions still requires manual work. To resolve this issue, we propose a solution to enrich the Web API descriptions by adding semantic annotations semi-automatically through API profiles that links properties to concepts in shared vocabularies. With this enriched descriptions, the compositions can be done either by directly linking the outputs and inputs of selected APIs, or by including transformation services that transform and make outputs compatible to inputs according to the semantic relations hold in the annotations. This work proposes an extension of the Open API Initiative (OAI) specification to create comprehensive descriptions and focuses on the emerging concept of API profiling to add descriptive information of data semantics by addressing Dublin Core Application Profile (DCAP) guidelines. One of our driving goals is to involve all groups of end-users in the Web API composition process. In the twenty first century, we are experiencing a rapid growth of elder people, who are using technologies for different purposes. Due to their different characteristics, the analysis of the context of elder domain has become a scientific challenge. Thus, we investigate the requirements, barriers and attitudes towards using systems or services of this group of people to understand the possibilities of transforming their roles from end-users to end-user developers. Moreover, we analyze different interface design guidelines to design a tool accessible by all groups of end-users including elders. We implement a prototype considering the underlying concepts and adopted design guidelines. Finally, we conduct an evaluation of the prototype using usability questionnaires.

ACKNOWLEDGEMENTS

I wish to thank all the people who contributed to this thesis work. At first, I would like to express my gratitude to my tutor Professor Stefania Bandini who gave me the opportunity to carry out my research work and supported me in every aspects. I would also like to thank Professor Carlo Batini for providing me the opportunity to work as a research assistant in the PASCH Project (Regione Lombardia-IBM-UniMiB research grant n.12A135).

It gives me immense pleasure to thank my mentor and supervisor, Professor Flavio DePaoli, for his support, continuous guidance, careful suggestions and intelligent criticism during the work phase and for his inexhaustible patience during the correction phase of this thesis. I appreciate all of his comments, input, suggestions and advice that have enabled me to accomplish this thesis in accordance with the timeline.

I am also grateful to Dr. Giuseppe Vizzari for his suggestions and guidance throughout this period of research. I am very much thankful to Christina Tziviskou for her sincere coordination and help during the first year of my PhD research. I would like to express my sincere gratitude to my colleague Marco Cremaschi for encouraging and coordinating me in my research works. I am very grateful to my colleague and friend Anisa Rula for her encouraging support and guidance. I am also thankful to Mamoun for his encouragement and prayers.

I acknowledge the support from Dr. Matteo Palmonari, Professor Lucia Pomello, Professor Gabriella Pasi, Blerina, Gorkem, Luca, Federico, Barbara, Antonio, Vincenzo and Pikakshi to me on various important occasions. I acknowledge and value the support of all researchers and staff of the ITIS Lab and the Department of Computer Science, Systems and Communications (DISCO) of the University of Milan-Bicocca for their support and cooperation throughout my doctoral study. I am also thankful to all the participants of the evaluation tests that have been conducted for this thesis.

I am highly indebted to Tunazzina apu, Jamal vai, Naveel and Naveed for their unconditional support, love and inspiration to me and my family during this journey. Moreover, I am grateful to the people who have always encouraged me during this difficult journey in Milan. My special thanks go to Shirin Apa, Faruk Bhai, Salim, Alauddin, Rakib, Shaffaet and many others. I am also thankful to my friends Shirin, Rosy, Suraiya and Shapla and my cousin Akter vai for always encouraging me to complete my thesis.

I believe that this small piece of work could not have been accomplished without the unconditional love of my family members, especially my mother Mrs. Jayeda Ali. For constant encouragement and support, my heartfelt gratitude goes to my sisters: Mahmuda Arzoo and Asma Shilpi, brothers: Anwar Hossain, Azad Hossain, Ruhul Amin and Saiful Islam and my sister-in-law Fatema Hossain, my nephews: Shafeen, Faris, Anas and nieces: Ariana, Sharfa for being there for me when I have needed someone to listen and care.

I deeply remember my father Mohammad Ali and my eldest brother Zahid Hossain who taught me to dream big and has been a great inspiration throughout my whole life. They are always with me in my thoughts and prayers. I am also thankful to every members of my in-laws family who always encouraged and supported me in every aspects specially my father-in-law Md. Golam Mustafa, brother-in-laws: Ahsan vai, Faisal and Bulbul.

Last but not the least, I must acknowledge the contribution and unconditional sacrifice of my husband, Md. Firoj Alam, who has always inspired and supported me to complete the tasks within the timeline. My love and affection go to my only daughter Liyana who has experienced the first-hand pain and pangs and tears and happiness of a doctoral student. Finally, I would like to dedicate this thesis to my husband and my daughter for being with me in spirit in every small step to reach this moment of thesis submission.

Author's declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmers and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

Milan, January 2017

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

This thesis presents a platform for user-driven Web API Composition by exploiting the end-user development approach. To accomplish this a proposal has been made considering the technological challenges to achieve the (semi) automatic composition of APIs that accelerate the whole process. Moreover, requirements of a specific group of end-users (i.e. elders) have been analysed to accommodate them in the end-user development environment with an easy to use tool. This work envisage to provide a way for the end-users to get a set of required information by composing related Web APIs to fulfil their specific needs. After summarizing the studies that motivate the requirements for creating this platform for the end-users, the chapter illustrates the research goals, research questions, main contributions and dissertation organization.

1.1 Motivation and Background

Over the last decade, the Web has grown from a large-scale hypermedia application for publishing and discovering documents (i.e., Web pages) into a programmable medium for sharing data and accessing remote software components delivered as a service or Application Programming Interface (API). The need of global availability and sharing of huge amount of information through various kinds of heterogeneous devices and services has changed the reference scenario for the development of Web scale applications [81]. In one hand, there is a plethora of Web services or Web APIs available in the registries (e.g. ProgrammableWeb[1]) often with generic and limited descriptions, on the other hand, there are complex needs of users that can not be fulfilled by accessing a single Web service or Web APIs. Therefore, there is a need to provide appropriate and

[1]http://www.programmableweb.com

complete Web service or Web API descriptions to let users discover services that satisfy a set of requirements and compose applications to fulfill more complex users' needs. Extensive research has been conducted with the vision to create automatic integration of Web services or APIs [120]. But in practice most of these approaches are having problems in communicating between candidate Web services or APIs due to the lack of semantic correlation of properties. To automate the interactions between Web services or Web APIs there is a need to describe the exchanged data with semantics. To support automatic composition of Web services, several approaches have been proposed focusing on semantic annotations, but many of them are either not validated or the validation lacks credibility [131]. Moreover, several description tools and formats have been introduced for describing REST Web APIs both in human and machine readable formats [88]. Although these descriptions provide functional information about the APIs (e.g. HTTP methods, URIs, model schema, etc.), the information that qualifies the properties of APIs (e.g. classification of input arguments and response data) is missing. These formats may meet the requirements of developers to complete simple tasks, but they are not enough in advanced API discovery or API composition due to the lack of machine processable semantics [136]. Moreover, such formats should be made easy to understand when the target users include high-level business experts or specific groups of people (e.g. the elderly, people with disabilities, etc.) who do not have specific programming expertise.

Existing literature shows that current challenges arise in the software engineering are the development of applications that can adapt to the heterogeneous needs of users [40] and shifting the role of end-users from mere consumers to becoming developers of applications addressing their unique, personal, and transient needs [4]. To resolve these issues, we adapt the end-user development approach [73] that is defined as "a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artefact".

The twenty-first century phenomenon known as the "demographic time bomb", refers to a dramatic increase in the number of elderly people and a reduction of younger people to care for them as they lose their independence in later years. According to the statistics presented in the literature [28, 133], the age structure of the world population is projected to dramatically change in the coming decades due to the dynamics of fertility, life expectancy and migration rates. This studies also show that 71% of them use computers at home and additionally, 62.4% access the Internet through a high-speed Internet connection. To face the challenge of this ageing structure the countries of Europe and US respond with the solution in investing in people to allow them to age actively. The active participation in the society and fulfil their own needs with new technologies is a challenge of this ageing phenomenon. We base our research on the premise that the revolution of web technology enables end-users to search for information according to their requirements, resulting older adults to maintain their independence and foster social inclusion. Although a number of research projects have sought to develop smart home environments to meet the special needs of the ageing population and offer them security, comfort and quality of life, initiative to make them independent in information seeking and utilizing these information to complete their daily needs is missing.

We also take into consideration that today's elder people are using technologies and services for different purposes [64]. Sometimes they need to get a set of information that can be provided by integrating different Web services and/or Web APIs. They are able to use new systems, applications or tools if those are provided with an easily accessible and simple to use platform. A issue here is what might appear as natural and intuitive to developers and designers may be confusing to the end-users. To resolve this issue, we analyse the requirements of a specific group of users (i.e., elders) to provide special supports they need while using the systems or tools [13, 64]. Although it is evident from the literature that non-programmer users are capable of using Web APIs (e.g. Mashup of APIs) successfully to fulfil their own needs [4], studies to accommodate a specific group of end-users (i.e. elderly) in the end-user development environment is missing. This stems our motivation to analyse the requirements of this group of end-users considering their age related changes and barriers. This proposal is envisioned to provide a platform where the tools and people are combined that embrace the shift from experts to everyone.

1.2 Research Goals and Research Questions

The research goal is to facilitate the (semi) automatic Web API Composition through a platform that considers the end-users' requirements. The particular research question is: *whether it is possible to provide an easy way to compose available Web APIs automatically according to user preferences*. To achieve this goal, this work is divided into two phases: one is resolving technological issues to support the Web API Composition at the back-end and another one is resolving the issues to support end-users including elders in executing the back-end processes through an easy to use interface. We consider the following questions related to composition to find out anticipated technological issues:

- How the Web services and Web APIs interact with each other?
- How to provide a comprehensive set of information to facilitate (semi) automatic composition?
- How the composition process can be easily accessible for all groups of end-users?

To analyse the above mentioned questions, we follow several literature such as [109, 120, 131, 136] and focus on the following important issues:

- Interoperability between heterogeneous Web services or Web APIs.
- Lack of a standard format for Web API descriptions to include a comprehensive set of information.

- Missing appropriate tools considering usability features for end-users.

In addition, for analysing the usability issues of end-users, we emphasis on understanding the needs and attitude of a specific group of end-users i.e. elders as a case study, considering the research question: *is it possible to exploit the opportunities of end-user development approach in a complex and an uninvestigated context (e.g. elderly)?*.

1.3 Main Contributions

To address the research questions this work has been divided into four phases: theoretical research and empirical analysis, design of novel solutions, implementation of prototypes, and evaluation or lesson learned. The initial phase of theoretical research and empirical analysis sheds light onto the potentials, the limitations, and the current state of the art in composition of Web services or Web APIs and end-user development approach. Based on the insights into technological issues, novel approaches are designed to facilitate user-driven (semi) automatic composition by providing RESTful interactions between involved Web Services or Web APIs and by enriching API descriptions through API Profiling and Semantic Correlation. The main contributions of this thesis are listed below:

- A requirement elicitation study analysing the opportunities of end-user development in a complex and an uninvestigated context (e.g. elder).
- A study on designing RESTful interactions between heterogeneous Web services or Web APIs to facilitate automatic composition.
- A framework that provides semantically enriched API descriptions that can correlate properties at the semantic level and facilitate (semi) automatic composition.
- A prototype to accommodate a specific group of end-users (i.e. elders) in the end-user development paradigm and enable them to compose Web APIs.

Our requirement elicitation study analyses the attitudes and needs of a specific group of end users, that is elders and illustrates their requirements to accommodate them in the end-user development paradigm.

Recent advances in radio-frequency identification (RFID) technology, near frequency communication (NFC) and sensor networks are fostering the emergence of environments where Internet applications and services are needed to be composed [135]. In particular, composition mechanisms in pervasive environments need to address context awareness, heterogeneity and contingencies of devices (e.g., unpredictable availability of services and mobile devices), and personalization (e.g., service provisioning based on user preferences). To resolve the technological issue of heterogeneity to foster interoperability, firstly a common framework for designing RESTful interactions between involved Web services and Web APIs has been designed by exploiting REST principles, HTTP methods and hypermedia controls. We consider sensors as services and design the Domain Application Protocol for describing the RESTful interactions of sensor services [81]. We also consider a case study to transform a SOAP based legacy Web service into a RESTful Web service by implementing a REST wrapper. During this implementation using JAX-RS and in-lab evaluation we found that it is very complex to update the existing SOAP Web services into RESTful Web services as server-clients are tightly coupled, thus inappropriate for clients when a service is changed. In addition, it requires extensive manual works to update existing services to support hypermedia controls that is one of the constraints of REST architectural style [30]. Other implementation related issues are also addressed like the internal ID in the database of the legacy service can not be used in the URL of RESTful Web service and for requests that requires the use of HTTP methods other than GET and POST, browser's object (XMLHttpRequest) needs to be created for sending HTTP requests instead of HTML forms. However, these complexities can be reduced by using new frameworks or libraries. We learned from this experiment that REST is a lightweight approach for developing Web services than SOAP Web services. Thus, we decide to use REST approach in our remaining works. As Web APIs are provided by third parties to access underlying resources, functionalities or data, through web-based user interfaces and programmatic interfaces of RESTful services are presented as REST APIs, we use the term Web APIs that follows the REST architectural style in rest of our research works.

To automate the process of Web API composition our proposal includes the enrichment of API descriptions by adding semantic annotations through API profiles that links properties to concepts in shared vocabularies. With this enriched descriptions, the compositions can be done either by directly linking the outputs and inputs of selected APIs, or by including *transformation* services that transform and make outputs compatible to inputs according to the semantic relations hold in the annotations. The *transformation* service has the task of managing the set of transformation rules that make properties compatible. This work proposes an extension of the Open API Initiative (OAI) specification[2] to create comprehensive descriptions and focuses on the emerging concept of API profiling to add descriptive information of data semantics by addressing Dublin Core Application Profile (DCAP)[3] guidelines. To accomplish this we propose to adopt a technique for adding annotation, TableMiner [150], which is a semantic table interpretation method to extract the most appropriate concepts from shared vocabularies in a (semi) automatic way by using context information. We conceived our approach to use existing vocabularies (about 558) indexed in the Linked Open Vocabularies search engine[4]. We anticipate the challenge to ensure the practicability with these limited number of vocabularies, but we rely on the statistics presented in [134] that shows domain independent vocabularies are covering different areas.

[2]http://openapis.org/specification
[3]http://dublincore.zsaorg/documents/profile-guidelines/
[4]http://lov.okfn.org/dataset/lov/

We believe additional domain specific vocabularies along with these vocabularies can be used to get a practical solution covering a large set of areas. The architecture and algorithms have been defined and a prototype considering the requirements of elders has been presented. In this work, we aim to provide an easy to use interface for the end-users including elders that hide the technical complexities from the end-user developers. At this point, the evaluation is done by using System Usability Scale (SUS) [11] testing method to determine the usability of this application. Although existing literature shows that elder people are using technologies for different purposes despite of having physical and cognitive barriers, inspiring them to compose services (i.e. Web APIs) by their own is a big challenge that has been encountered during this evaluation phase.

These contributions help to bridge the gap between users' requirements and available Web APIs, hence support end-users in the creation of their desired applications to fulfill their specific needs. The work is also useful for extending end-user development potentials into all kinds of end-users including elders. By extending the OAI specification for creating a comprehensive API description format and adding semantic annotation to correlate properties of involved Web APIs, this work facilitates the (semi) automatic composition at the back end. Moreover, our designed interface provided in the prototype is easily accessible and facilitates all kinds of end-users including specific group of people (i.e. elders).

1.4 Dissertation Organization

This chapter provides the motivation behind our research as well as a short introduction to our main contributions. The remainder of this dissertation is structured as follows:

Chapter 2 introduces the reader to the basic concepts and technologies necessary for understanding this thesis. It includes a short overview of the architecture of the World Wide Web as well as an introduction to the Services on the Web, Service Descriptions, Semantic Web Technologies, Web API Profiling and Service Composition . Finally, the chapter discusses users' role in the end-user development paradigm and classifies them into two main categories, namely *Professional Developers* and *End-user Developers*.

Chapter 3 distills shortcomings and issues from the current best practices for the creation, documentation, and composition of Web services or Web APIs. It looks at the difficulties in composition of Web services or APIs and adoption of Web APIs in the end-user development paradigm. It also discusses the current Web API Description formats that lack in providing a comprehensive set of information to both the human users and machine agents. Lastly, the chapter formalizes the research problems addressed by this thesis.

Chapter 4 reviews recent related research works that have been conducted in the area of end-user development approaches, RESTful composition and Web service descriptions. It also presents a number of interface description languages and Open API Initiative specification that are relevant for this thesis and discusses the open issues.

Chapter 5 describes the context of this thesis. We select a specific group of end-users (i.e elders). We look at the needs of this specific group of end-users and discuss their attitudes towards using new systems or technologies to understand their requirements. Finally, the chapter addresses the requirements of elders for interacting with the Web API composition tool, considering the HCI guidelines for elders and design for all perspective.

Chapter 6 describes the RESTful interactions between Web services or Web APIs to facilitate automatic composition. The description of this solution begins with an explanation of its basic concepts and principles and is followed by an illustrative example showing how it might be used in practice. It considers sensors as services and looks at the adoption of REST constraints in RESTful interactions by employing an uniform interface and using HTTP methods. It also discusses the Domain Application Protocol for sensor services. Moreover, this chapter evaluates the practicality of this solution and discusses the challenges the developers faced during the implementation.

Chapter 7 discusses the addition of API profiling in Web API descriptions to provide a comprehensive set of information that facilitates the (semi) automatic composition of Web APIs. We extend the Open API Initiative (OAI) specification for creating Web API descriptions and adding semantic annotation to facilitate composition by mapping semantic correlation of properties. The presented architecture describes how the system works. This chapter also discusses the composition process using the semantically enriched Web API descriptions.

Chapter 8 presents our solution for user-driven Web API composition that set out a bridge to reduce the gap between the end-users' requirements and technology constraints. This chapter describes the prototype that considers the easy to use interface for all groups of end-users including elders considering the requirements pointed out in chapter 5. Finally, it presents the evaluation results of the developed prototype by adopting System Usability Scale (SUS) testing method and involving all groups of end-users.

Chapter 9 concludes the thesis by briefly revisiting and summarizing the main findings and contributions, identifying limitations of the proposed solutions, and discussing future research directions. Some of the relevant findings and contributions have already been published in conference proceedings.

Chapter 2

Basic Concepts and Technologies

Today's world without Internet and World Wide Web is almost unimaginable. Although the terms Internet and World Wide Web are often used interchangeably in everyday speech, it is technically incorrect. The term Internet refers to the global system of interconnected computer networks. It is a network of networks using the Internet protocol suite; commonly known as TCP/IP due to the Transmission Control Protocol (TCP) and the Internet Protocol (IP) the first two protocols defined back in 1981 [70]. On the other hand, the World Wide Web is just one of the many applications running on the Internet. Today it is by far the most popular application on the Internet and overtook other applications such as file transfer or newsgroups many years ago. The Web is an information system of interlinked hypertext documents, so called web pages, that Tim Berners-Lee proposed in 1989 to build a more efficient internal information system. After realizing the potential of this approach, it has been used globally across organizations and became an important part of the world. According to statistics[1], around 40 percent of the world population has an internet connection today. In 1995, it was less than 1 percent (see Fig. 2.1). The number of internet users has increased tenfold from 1999 to 2013. The first billion was reached in 2005, the second billion in 2010 and the third billion in 2014. Fig. 2.1 shows the number of global internet users per year since 1993.

Considering its sheer size, its often uncoordinated development, and its heterogeneity, the Internet and the World Wide Web are the most complex systems ever built by humankind. In the following sections, we will have a look at the architecture that enabled the Web's exponential growth and some of the main technologies related to User-driven Web API composition. We will provide an overview of related technologies such as Web Services, Web APIs, REST, Web Service Composition, Semantic Web and End-user development. In the context of this thesis, the

[1] http://www.internetlivestats.com/internet-users/

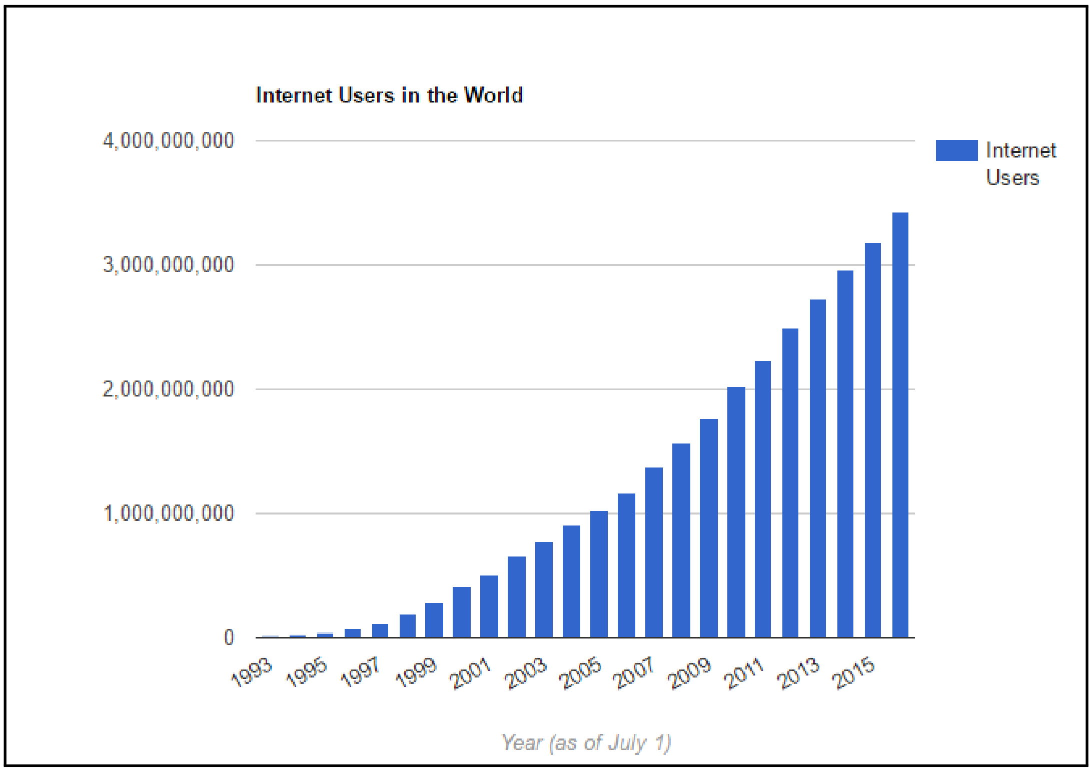


Figure 2.1: Internet users in the world (source:http://www.internetlivestats.com/internet-users/)

underlying Internet technology is assumed as a given infrastructure and thus not being discussed in detail.

2.1 Architecture of the World Wide Web

The World Wide Web[2] uses relatively simple technologies with sufficient scalability, efficiency and utility that they have resulted in a remarkable information space of interrelated resources, growing across languages, cultures, and media. Web can be described as a giant, globally distributed collection of hypertext documents where links and the resulting networking effects played a fundamental role for the success of the Web. Universal Resource Identifiers provide a mechanism to enrich documents with references to other relevant documents. The key differentiator to other hypertext systems available at the time lies in the decision to make links unidirectional instead of requiring them to be bidirectional. Practically, this means that it is impossible to prevent broken links when documents become unavailable. While link rot is certainly an undesired consequence of this decision, the advantages clearly outweigh this shortcoming. The decision is arguably one of the main reasons for the Web's superior scalability. As unidirectional links eliminate the otherwise necessary referential integrity checks, it is possible to drastically simplify

[2]https://www.w3.org/TR/webarch/

the implementation of clients and servers. Furthermore, unidirectional linkage enables the decentralized, uncoordinated creation of documents. This decentralism was the main reason why the Web quickly overshadowed all previous hypertext systems and allowed it to scale in an unprecedented manner. For a long time, the Web's architecture and its technologies were not properly standardized. The documentation merely consisted of a set of informal web pages, draft specifications, and the source code for clients and servers published by CERN [57]. As those documents have not been kept in sync with the deployed implementations, it became harder and harder to create interoperable systems. Consequently, the pressure from industry asking for standardization of the core technologies grew and working groups writing stable specifications were established. After working for several years, officially standardized syntax for Uniform Resource Identifier has been published where a document has been identifies as a "resource". A resource can be anything that has identity. Familiar examples include an electronic document, an image, a service (e.g., "today's weather report for Milan"), and a collection of other resources. Not all resources are network "retrievable"; e.g., human beings, corporations, and bound books in a library can also be considered resources. The resource is the conceptual mapping to an entity or set of entities, not necessarily the entity which corresponds to that mapping at any particular instance in time. Thus, a resource can remain constant even when its content, the entities to which it currently corresponds changes over time, provided that the conceptual mapping is not changed in the process. In 1997, the first normative specification of the Hypertext Transfer Protocol has been published. In the mid-nineties, Roy T. Fielding, co-author of both the URI and the HTTP specification, started working on "an architectural model for how the Web should work, such that it could serve as the guiding framework for the Web protocol standards" [30]. The outcome of his work was an architectural style that he called Representational State Transfer (REST). According to him, REST captures all aspects of a distributed hypermedia system that are considered central to the behavioral and performance requirements of the Web. Since REST has been used to guide the design and development of the architecture of the modern Web [31], we will discuss it in more detail in the next section.

2.2 Services on the Web

Many pages on Web are built by segmenting and flattening structured data from databases to HTML documents. To process the data published in such a way, brittle approaches such as screen scraping have to be used to at least partially reconstruct the raw data in order to make it machine processable. The aim of the Semantic Web is to eliminate this limitation by creating a Web of Data that can be directly processed by machines. In practice, however, that goal is still rarely approached by using Semantic Web technologies. Instead, the majority of structured data is being published in the form of XML[3] or JSON[4] documents. Commonly speaking, such offerings are

[3]https://www.w3.org/XML/
[4]http://www.json.org/

termed as Web services but given that also human-targeting, HTML-based websites are, at least to a certain degree, machine-processable, that term is somewhat misleading as websites could be classified as Web services as well. Similarly, a Web service could be referred to as a "website for machines". Thus, for the scope of this dissertation, we consider the term Web service as a set of HTTP-based interfaces to support interoperable machine-to-machine interaction by the exchange of structured data as defined in [70]. The aim of the machine-to-machine interaction is to drive business processes in order to serve particular, application dependent goals. Since Web services put the emphasis on "machine-to-machine", the interactions are optimized for machines instead of being optimized for humans as websites are. In practice, two major classes of Web services can be identified, namely SOAP-based services and RESTful services. We will discuss both in the following sections.

2.2.1 SOAP based Services

In an effort to improve the flexibility and dynamicity of their products, the information technology industry started to work on the formalization and standardization of Web services in the late nineties. The outcome was a complex set of specifications. XML was, mainly due to its popularity at the time, chosen as the main data format. The other three main pillars are SOAP, WSDL, and UDDI. SOAP [39], originally defined as Simple Object Access Protocol at Microsoft, specifies a messaging framework consisting of a processing an extensibility model, a protocol binding framework as well as a XML-based message format. The Web Service Description Language (WSDL) [20] describes the interface of a Web Service and Universal Description, Discovery and Integration (UDDI) registries that allow the discovery of services and their interface descriptions. Even though huge investments have been made, the promise of uniform service interface standards and universal service registries in the form of SOAP, WSDL, and UDDI has proven elusive. The Universal Business Registry, the main public UDDI registry, has been shut down in 2006 and most public SOAP-based Web services have been phased out shortly thereafter. There were many problems that led to this demise but the most fundamental reason is that the whole architecture is based on a Remote Procedure Call style which has been known to be flawed for years [140]. Furthermore, instead of using HTTP as an application protocol, it was misused as a transport protocol. In SOAP, e.g., data is typically retrieved by POSTing a SOAP-request to the service which then returns the desired data. This breaks intermediaries that serve as proxies or caches which typically perform their functions based on the standard semantics associated with the HTTP verbs and headers in the messages flowing through them. In practice, this reduces the scalability enormously and means that running a public service often results in prohibitive costs. Consequently, the number of public SOAP-based services has become negligibly small. Looking at company-internal services, however, the extensive tooling often outweighs these disadvantages so that they are still used in such scenarios. Another problem when there is no coordination between the publisher and the consumer of a service is that, despite using abstractions of the data types found in the actual implementations, services interfaces described with WSDL often leak implementation details which leads to tightly coupled systems. Similarly, the mapping to the abstract types is not always easily possible and thus often results in severe interoperability problems. Especially the inherent impedance mismatch between XML Schemas (XSD) and object-oriented programming constructs, the so called O/X impedance mismatch, often complicate the integration of different systems. The XML Schema language has a number of type system constructs which simply do not exist in commonly used object-oriented programming languages such as Java. Thus, interoperability problems arise because each SOAP stack has its own way of mapping the various XSD type system constructs to objects in the target platform's programming language and vice versa. In summary, the problems outlined above and the complexity of the technology, which consists of far more specifications than just SOAP, WSDL, and UDDI, led to a shift towards more lightweight solutions that integrate better into the architecture of the Web, which we will describe in the next subsection.

2.2.2 Services based on REST Architectural Style

In his 2000 dissertation [30], Roy Thomas Fielding coined the term REST. While often misunderstood, his dissertation has nonetheless been very influential. As of today, there exist an abundance of Web APIs that claim to be RESTful, even though most of them don't adhere to all the criteria listed by Fielding. In his dissertation, Fielding analyzed the architecture of the World Wide Web and generalized it to an architectural style for network-based application software. An architectural style is a named, coordinated set of architectural constraints. He called the architectural style as Representational State Transfer (REST). As introduced by Fielding, REST is an architectural style where the system adheres to the constraints listed below. They are squarely aimed at making REST suitable for Internet-scale distributed systems, which come with a unique set of challenges such as potentially heavy load (e.g. on a single resource), latency, and intermittent network failures. The constraints are:

1. There are distinct clients and servers.
2. The system implements a stateless communication protocol between the client and the server, "such that each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client". This allows servers to free up resources, such as memory, after each request and thus communicate with many clients simultaneously with relatively little burden placed on the server. This is in contrast to architectural styles such as RPC which are less suited for the Internet-scale loads.
3. The server can label responses as cacheable or non-cacheable. The client is free to reuse the response of cacheable responses in the future instead of sending the same request again.

4. “The central feature that distinguishes the REST architectural style from other network-based styles is its emphasis on a uniform interface between component”, which includes the following:

   - “The key abstraction of information in REST is a resource.” Anything that can be assigned a URL can be a resource: a document or image, a temporal service , a collection of other resources [or] a non-virtual object (e.g. a person). Note that REST isn’t constrained to CRUD-style APIs: also services, which may do a significant amount of computation on the server, can be resources.
   - When a URL is dereferenced, the server sends a resource representation to the client. Note that the resource representation (e.g. the HTML or JSON describing a person or a text document) is different from the conceptual resource itself (e.g. the person or text document). The client ever only accesses the representation, and manipulation of the resources is always done indirectly via the representation. By making explicit the step of serializing the resource to send its representation over a network, instead of working with the leaky abstraction or illusion that the client is working directly on the server’s resource, helps to deal with various scenarios, like: network errors, race conditions when multiple clients work on the same resource on the server, and idiosyncrasies in the serialization (and deserialization) of the resource to (and from) its representation. Furthermore, the serialization format or “data format of a representation is known as a media type”. HTML, JPEG, XML and JSON are all examples of media types. The specification of the data format, which the client has to be familiar with in advance, generically informs the client on how to construct further operations upon receiving a resource representation. The media type, however, does not specify the structure/schema of the resource itself.
   - HATEOAS (Hypermedia as the Engine of Application State) means that the media type used should be a hypermedia format(i.e.it should at least have the notion of a link) and that the client requires no prior knowledge beyond a generic understanding of the hypermedia format to interact with the server. Similar to how a web browser only needs to know HTML and the URL of the homepage of a website, an API client only has to know a supported media type and the URL of an entry point to the API. It can then follow links to dynamically walk the API’s resources. With each HTTP request, the state of the application, as expressed by the client’s current URL, is changed. No further out-of-band information, such as schemas or API descriptions, is needed. A hypermedia format has to define a way to construct valid state transitions from a server response. For example, the HTML format specifies how to construct a GET request from a link element and its href attribute, as well as how to construct a POST request from a form and its containing elements. HATEOAS enables very loose coupling between the client and the server. Also, just as a web browser can navigate

every website using HTML, generic clients can interact with all APIs that talk in a hypermedia format supported by the client.

5. The system must be a layered system with components constrained “such that each component cannot see beyond the immediate layer with which they are interacting”. This allows the clients to communicate transparently with proxies and gateways. For example when a caching proxy is introduced, the client can just talk to the new proxy as if it were the original server, although the proxy actually passes the request on to the next layer behind the scenes.

6. The system may use code on demand. The server may not only send data but also code to the client where additional processing would take place. In terms of the web, that would be JavaScript or Java applets executed on the client.

According to the literature [81, 109, 152], some of the advantages of RESTful Web services include:

Light-weight: RESTful Web services directly utilizes HTTP as the invocation protocol which avoids unnecessary XML markups or extra encapsulation for APIs and input/output. The response is the representation of the resource itself, and does not involve any extra encapsulation or envelopes. As a result, RESTful Web services are much easier to develop and consume than WSDL/SOAP Web services, especially in the Web 2.0 context. Additionally, they depends less on vendor software and mechanisms that implements the additional SOAP layer on top of HTTP. RESTful Web services usually deliver better performance due to their light-weight nature.

Easy-accessibility: URIs used for identifying RESTful Web services can be shared and passed around to any dedicated service clients or common purpose applications for reuse. The URIs and the representation of resources are self-descriptive and thus makes RESTful Web services easily accessible. RESTful Web services have been widely used to build Web 2.0 applications and mashups.

Scalability: The scalability of RESTful Web services comes from its ability to naturally support caching and parallization/partitioning on URIs. The responses of GET (a side effect free operation) can be cached exactly the same as web pages are currently cached in the proxies, gateways and content delivery networks (CDNs). Additionally, RESTful Web services provide a very simple and effective way to support load balancing based on URI partitioning. Compared to ad hoc partitioning of functionalities behind the SOAP interfaces, URI-based partitioning is more generic and flexible,and could be easier to realize.

Declarative: In contrast to imperative services from the perspective of operations, RESTful Web services take a declarative approach and view the applications from the perspective of resources. Being declarative means that RESTful Web services focus on the description of the resources themselves, rather than describing how the functions are performed. Declarative style brings the fundamental differences between RESTful Web services and WSDL/SOAP Web services to the forefront. While building services for a particular system, the declarative approach focuses on what resources needed to be exposed and how these resources can be represented; imperative approach focuses on what operations needed to be provided and what are the input/ouput of these operations. Declarative approach is considered to be a better choice to build flexible, scalable and loosely-coupled SOA systems.

According to statistics from ProgrammableWeb[5], the premier catalog of public Web services, three out of four Web services are based on the REST architectural style. This does not mean that they fully obey REST constraints but primarily that they do use HTTP as an application protocol and that the various resources get their own IRIs. In fact, most services that claim to be RESTful (REST APIs) are not. To capture the various levels of "RESTfulness", Richardson defined a maturity model [107] but by definition, a service is either RESTful by obeying to all constraints defined by REST or it is not. Annoyed by the fact that especially the hypermedia constraint is ignored by many Web services that claim to be RESTful, Fielding wrote a blog post [31] making it clear that the constraint it is not optional. Since the term REST was so often misused, recently HTTP-based Web services are typically simply referred to as Web APIs instead. For services that do obey to REST's hypermedia constraint, the term Hypermedia API has become popular.

Web APIs typically have in common that they use a very small set of standards. Most often, it consists of just HTTP and either XML or JSON as the serialization format. Even though XML with its name spacing support that allows messages to be enriched with hypermedia controls or to be made self-descriptive would be better suited for RESTful services, JSON has become the preferred data interchange format for Web APIs in recent years. The downside of this simplicity is that most services are unique and only documented in natural language. This renders automatic code generation or the creation of generic tooling almost impossible.

2.3 Web Service Composition

In recent years, Web services have received much interest as an emerging technology for Business-to-Business Integration (B2Bi) of heterogeneous applications over the Internet. Many enterprises are transforming their business model in Internet with Web services because Web services technology enables integration of heterogeneous applications regardless of implementation platforms. The full potential of Web services as a means for B2Bi solutions will only be realized when existing services and business processes are able to integrate into a value-added composite service. A composite service is a service developed by aggregating the existing services to realize a new value-added functionality. The aggregating process is called (Web) service composition. There are two approaches of Web service composition: static and dynamic composition. In static compositions, the aggregation of the services is done at design time. The composition of all the necessary components is determined, bound together and then deployed. This type of composition

[5]http://www.programmableweb.com

is more suitable if the business partners involved in the process are fixed and their offered functionalities or the requirements are unlikely to change in short term. Static Web service composition is not flexible in the sense that it is not adaptive to the changes when it is in execution and is too restrictive to unavoidable changes in the service requirements. Dynamic Web service composition aims at overcoming the problems which are apparent in static Web service composition. A dynamic service composition provides flexibility for modifying, extending and adapting changes at run time. The Web service environment is highly dynamic in nature. The number of web service providers is constantly increasing leading to the availability of new services in daily basis. In such a dynamic environment, realizing dynamic web service composition is not so easy because of the following reasons:

- composition process should discover the appropriate components in a composition that is able to transparently adapt the environmental changes.
- composition process should adapt to the customer requirements with minimal user intervention.
- composition process should have transactional support.
- composition process should guarantee composition correctness.
- composition process should also consider Quality of Service (QoS) properties.

Because of the growing trend of transforming existing business models to the Internet with Web services and the possibility of creating new Web services dynamically, various developments are on-going in the dynamic web service composition [72]. Moreover, Mashup is essentially a process that integrates data/content from different sources on the internet. Therefore, it is essentially a service composition style from SOA perspective. Web services composition allows organizations to form alliances, to outsource functionalities, and to provide one-stop shops for their customers. From a business perspective, services composition dramatically reduces the cost and risks of building new business applications in the sense that existing business logics are represented as Web services and could be reused [63].

2.3.1 Comparison between Mashup and Service Composition

Mashup is a Web application that combines data from multiple data sources into a single integrated application. A mashup site must access third party data using APIs, and should add value to these data during the integration. Data could come from local databases or various sources across the Internet via different protocols including HTTP, RSS, ATOM and RESTful Web Services. Compared to RESTful Web service composition, a mashup is restricted at the data-level integration, and most uses of RESTful Web Service in mashup are limited to fetching data from remote sources [152]. It usually does not involve updating or manipulating remote data sources or other resources.

As mashup is relatively a new concept, it may be confusing to distinguish mashup and traditional service composition. Mashup is essentially a process that integrates data/content from different sources on the internet. Therefore, it is essentially a service composition style from SOA perspective [74]. Obviously, the mashup is the extension of current SOA paradigm. It does not break the basic SOA principles (services are self-described and loosely coupled) while making services more easily and visually utilized by consumers. The core issue here is the mashup component. Based on traditional service concept (such as Web services), mashup component enables the interoperability between different service providers at UI level. Thus, mashup is mainly the composition at UI level, while current composition at logic level.

2.4 Semantic Web and Linked Data

The early standardization efforts of the Web made it clear that it is more than a hypertext document system. The separation of documents into resources and representations thereof paved the way to integrate real world entities such as persons or even imaginary or abstract concepts such as companies into the Web at an architectural level. Tim Berners-Lee felt urged to express his vision more explicitly to a wider public at the First International World Wide Web Conference in 1994 and he argued that the Web has become an "exciting world" for users but that it contains very little machine-readable information. "The meaning of the documents is clear [only] to those with a grasp of (normally) English, and the significance of the links is only evident from the context around the anchor [but to a computer] the web is a flat, boring world devoid of meaning" [6]. To add machine-readable semantics to the Web, two necessary things had been identified by him, (i) allowing documents to contain machine readable information and (ii) allowing links with explicit relationship values. Unsurprisingly, the World Wide Web Consortium (W3C), which Berners-Lee founded later that year to coordinate the standardization of the Web, put a focus on standardizing Semantic Web technologies. After various related efforts, this resulted in the standardization of the Resource Description Framework (RDF) in 1999 [71].

The Semantic Web is an extension of the traditional Web with the aim to offer information not only in the form of natural language documents but also as machine-readable data. The Resource Description Framework (RDF)[6] builds the foundation of the Semantic Web technology stack. RDF is a standard model for data interchange on the Web. It defines a simple, triple-based data model in which each statement consists of a subject, a predicate, and an object. Multiple triples build a graph, and multiple graphs form a data set. While Internationalized Resource Identifier (IRIs) can be used in every element of an RDF triple, literals, i.e., basic values such as strings or numbers which are typed and optionally language-tagged, can only be used in the

[6]https://www.w3.org/RDF/

object position. Blank nodes, which are special local identifiers whose scope is limited to a single document or data store, can only be used in the subject and the object position; not as predicates. In 2006, however, Shadbolt, Hall, and Berners-Lee published an article [119] admitting that the simple idea behind the Semantic Web vision still remained largely unrealized. The few early Semantic Web projects lacked viral uptake and only very few used dereferenceable URIs which would have allowed the data to be browsed in a similar fashion as documents can be browsed on the Web. This stems to formulate the famous Linked Data principles [7] which can be classified as a turning point in the history of the Semantic Web.

Due to the fact that the early Semantic Web could not be browsed like the traditional Web, it became a separate ecosystem rather than an extension of the Web. All the technologies that form the Semantic Web technology stack have in common that they do not require IRIs to be dereferenceable. Instead, just as RDF itself, they treat them as opaque identifiers. In an effort to make it works like web, Tim Berners-Lee postulated the following Linked Data principles in 2006 [7]:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
4. Include links to other URIs. So that they can discover more things.

These four simple principles represent an important turning point in the history of the Semantic Web. Not only did they rebrand the vision of a Semantic Web with a much more concrete and graspable description of the basic principles underlying it but also refocused its applications on more practice-relevant problems, namely the publication and consumption of vast amounts of structured data.

2.4.1 Vocabularies and Ontologies

The word ontology comes from philosophy, which means "the Knowledge of what is to be in one self" [7]. In data processing, ontology indicates a structured set of Knowledge in a domain of interest. Ontology is conceptualization of an application domain in a human-understandable and machine-readable form, and typically comprises the classes of entities, relations between the entities and the axioms which apply to the entities which exist in that domain. Several operations are possible on ontology, such as Mapping, Matching, Alignment, Transformation, Merging and Integration etc. On the Semantic Web, vocabularies define the concepts and relationships (also referred to as "terms") used to describe and represent an area of concern. Vocabularies are used to

[7] http://www.daml.org/ontologies

classify the terms that can be used in a particular application, characterize possible relationships, and define possible constraints on using those terms. In practice, vocabularies can be very complex (with several thousands of terms) or very simple (describing one or two concepts only). There is no clear division between what is referred to as “vocabularies” and “ontologies”. The trend is to use the word “ontology” for more complex, and possibly quite formal collection of terms, whereas “vocabulary” is used when such strict formalism is not necessarily used or only in a very loose sense. Vocabularies are the basic building blocks for inference techniques on the Semantic Web. The role of vocabularies on the Semantic Web are to help data integration when, for example, ambiguities may exist on the terms used in the different data sets, or when a bit of extra knowledge may lead to the discovery of new relationships. Consider, for example, the application of ontologies in the field of health care. Medical professionals use them to represent knowledge about symptoms, diseases, and treatments. Pharmaceutical companies use them to represent information about drugs, dosages, and allergies. Combining this knowledge from the medical and pharmaceutical communities with patient data enables a whole range of intelligent applications such as decision support tools that search for possible treatments; systems that monitor drug efficacy and possible side effects; and tools that support epidemiological research. Another type of example is to use vocabularies to organize knowledge. Libraries, museums, newspapers, government portals, enterprises, social networking applications, and other communities that manage large collections of books, historical artifacts, news reports, business glossaries, blog entries, and other items can now use vocabularies, using standard formalisms, to leverage the power of linked data. It depends on the application how complex vocabularies they use. Some applications may decide not to use even small vocabularies, and rely on the logic of the application program. Some application may choose to use very simple vocabularies like the one described in the examples section below, and let a general Semantic Web environment use that extra information to make the identification of the terms. Some applications need an agreement on common terminologies, without any rigor imposed by a logic system. Finally, some applications may need more complex ontologies with complex reasoning procedures. It all depends on the requirements and the goals of the applications. To satisfy these different needs, W3C offers a large palette of techniques to describe and define different forms of vocabularies in a standard format[8]. These include RDF and RDF Schemas, Simple Knowledge Organization System (SKOS), Web Ontology Language (OWL), and the Rule Interchange Format (RIF). The choice among these different technologies depend on the complexity and rigor required by a specific application. A general example may help. A bookseller may want to integrate data coming from different publishers. The data can be imported into a common RDF model, eg by using converters to the publishers’ databases. However, one database may use the term “author”, whereas the other may use the term “creator”. To make the integration complete, and extra definition should be added to the RDF data, describing the fact that the relationship described as “author” is the same

[8]https://www.w3.org/standards/semanticweb/ontology

as “creator”. This extra piece of information is, in fact, a vocabulary (or an ontology), albeit an extremely simple one. In a more complex case the application may need a more detailed ontology as part of the extra information. This may include formal description on how authors are to be uniquely identified (e.g. in a US setting, by referring to a unique social security number), how the terms used in this particular application relate to other datasets on the Web (e.g. Wikipedia or geographic information), how the term “author” (or “creator”) can be related to terms like “editors”, etc.

2.5 Service Descriptions

Service descriptions contain information on the functionalities and the data provided by web services, possibly in a machine understandable way [127]. Service descriptions are used for service consumption, documentation, testing, or automatic generation of client (or server) code in various languages. Today, most web services expose web application programming interfaces (Web APIs), where Web API descriptions define the set of possible interactions with a web service. While for SOAP services, WSDL is the largely agreed-upon standard, various description formats have been proposed for services exposing web APIs, including the web application description language (WADL)[9], Swagger[10], or RAML[11] . These formats differ in their focus on describing either data and/or functionalities and in their intended usage by either humans or computers. These descriptions include implementation details such as resources and URLs, representation formats (HTML, XML, JSON, etc.), status codes, and input arguments in both a human and machine-readable form. In our work, we consider descriptions written in machine-understandable languages such as XML or JSON.

2.5.1 Semantic Annotation

An annotation assigns to an entity, which is in the text, a link to its semantic description. A semantic annotation is referred to an ontology. The idea is to have data through the Web defined and linked in such a way that its meaning is explicitly interpretable by software processes rather than just being implicitly interpretable by humans. Semantic annotation can be applied to any resource (file, image, Web page, etc.) [90].

2.6 Web API Profiling

A more recent development in API meta services is the idea of creating and sharing API Profile information. Unlike description documents, profile documents offer a high-level view of what

[9]http://www.w3.org/Submission/wadl/
[10]https://www.swagger.io
[11]http://raml.org/

the API supports and, in some cases, how clients and servers can expose features in a machine-readable way. The HTML 4.01 specification[12] introduced the profile attribute in 1999. The Meta data profile was defined as either a) a globally unique name (URI) or, b) a link (URL) to an actual document. It was designed to allow document authors to provide additional descriptive information about the contents of the response (e.g. useful indexing properties of the document, terms of use, etc.).

One of the foundations of the Internet and web architecture is the fact that resource representations communicated through protocols, such as SMTP or HTTP, are labeled with a 'media type', which allows a client to understand at run time what 'type' of resource representation it is handling[13]. Sometimes, it would be useful for servers and clients to include additional information about the nature of the resource. This would allow a client understanding this additional information to react in a way specific to that specialization of the resource, where the specialization can be about constraints, conventions, extensions, or any other aspects that do not alter the basic media type semantics.

Although the concept of a profile has no strict definition on the Internet or on the web, the term profile has been widely used to refer to a document that describes how standards or specifications are deployed to support the requirements of a particular application, function, community or context, and the term application profile has been applied to describe this tailoring of metadata standards by their implementers [49]. In general, a profile is described as additional semantics that can be used to process a resource representation, such as constraints, conventions, extensions, or any other aspects that do not alter the basic media type semantics. A profile must not change the semantics of the resource presentation when processed without profile knowledge, so that clients both with and without knowledge of a profiled resource can safely use the same representation. Profiles can be combined, meaning that a single resource representation can conform to zero or any number of profiles. Depending on the profile support of clients, it is possible that the same resource representation, when linked to a number of profiles, can be processed with different sets of processing rules, based on the profile support of the clients.

The Dublin Core Metadata initiative (DCMI)[14] has published a formalization of the Dublin Core metadata model called the DCMI Abstract Model[104], which provides the necessary foundation for a formalization of application profiles that lends itself to machine processing.

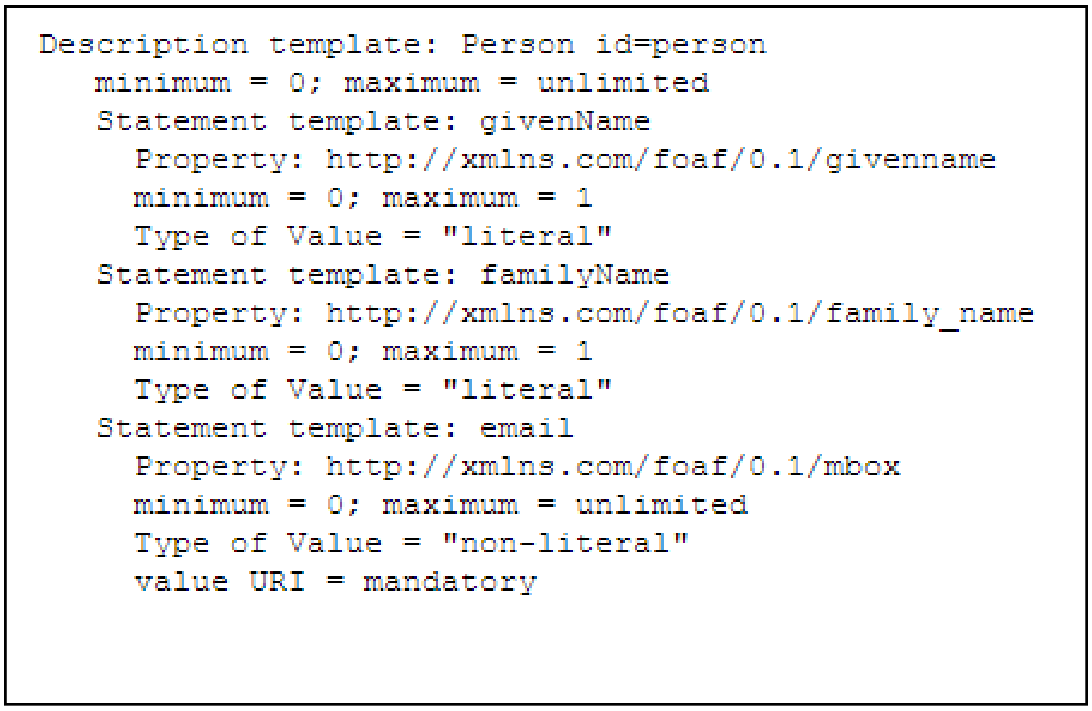
2.6.1 Metadata Profile

Considering the fact that there are various metadata needs of particular communities and applications due to their diversity, Dublin Core Metadata Initiative (DCMI) provided a framework for designing a Dublin Core Application Profile (DCAP). A DCAP defines metadata records which

[12]https://www.w3.org/TR/html4/

[13]https://www.ietf.org/rfc/rfc6906.txt

[14]http://dublincore.org/



```
Description template: Person id=person
   minimum = 0; maximum = unlimited
   Statement template: givenName
     Property: http://xmlns.com/foaf/0.1/givenname
     minimum = 0; maximum = 1
     Type of Value = "literal"
   Statement template: familyName
     Property: http://xmlns.com/foaf/0.1/family_name
     minimum = 0; maximum = 1
     Type of Value = "literal"
   Statement template: email
     Property: http://xmlns.com/foaf/0.1/mbox
     minimum = 0; maximum = unlimited
     Type of Value = "non-literal"
     value URI = mandatory
```

Figure 2.2: Description template of a Person following DCAP guidelines

meet specific application needs while providing semantic interoperability with other applications on the basis of globally defined vocabularies and models[15]. A DCAP is a generic construct for designing metadata records that does not require the use of metadata terms defined by DCMI. A DCAP can use any terms that are defined on the basis of RDF, combining terms from multiple namespaces as needed. A DCAP follows the DCMI Abstract Model (DCAM), a generic model for metadata records. A DCAP includes guidance for metadata creators and clear specifications for metadata developers. By articulating what is intended and can be expected from data, application profiles promote the sharing and linking of data within and between communities. The resulting metadata is integrated with a semantic web of linked data. The metadata record's design is detailed in a Description Set Profile (DSP) that contains one Description Template for each thing in the domain model, which in turn contain statement templates for all of the properties that describe the thing. These templates also define any rules that constrain the use of the description or the properties, such as value types or requirements and repeatability.

As shown in 2.2, a Person can have one optional given name and one optional family name, each of which are literal strings. A Person can also have an email address which must be represented by a URI with the prefix *mailto:*. Because many of us have more than one email address, we allow this statement to repeat as often as necessary. The *Property: http://xmlns.com/foaf/0.1/givenname* links to the Semantic Web vocabulary[16] of givenName. FOAF is devoted to linking

[15]http://dublincore.org/documents/profile-guidelines/
[16]http://xmlns.com/foaf/spec/

people and information using the Web. Regardless of whether information is in people's heads, in physical or digital documents, or in the form of factual data, it can be linked. FOAF descriptions are themselves published as linked documents in the Web (eg. using RDF/XML or RDFa syntax).

2.7 End-User Development

According to [73], End-User Development can be defined as "a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artefact". Traditionally, software engineering distinguished between programmers (or developers) and end-users, the former developing software, the latter using it. Over the last years, however, these roles have increasingly blurred, and today this distinction is no longer as clear as one might think. Different roles have been introduced by the End-user development research works [4, 14, 37]. In the course of our analysis, we identified the following three types of roles of users (see Fig. 2.3):

1. Professional Developers: Professional developers have all the necessary composition skills to develop also very sophisticated composites, possibly including solutions to crosscutting concerns like transactions or security. They have sufficient knowledge of the necessary language notations and composition paradigms and, if not, know how to acquire such autonomously. One key skill of professional developers is the ability to develop new APIs or components for reuse by others, not only composite applications. Programmers developing RESTful Web services, SOAP Web services, UI widgets, and so on fall into this category, as do programmers using composition languages like BPEL or directly Java or programming languages. They not only develop simple composites but also complex, mission-critical B2B integrations.

2. End-user Developers: These are programmers that are able to develop their own "computations" in the form of composite applications that serve some limited, typically personal, purpose. They know about components, software reuse, and are familiar with some composition paradigm that allows them to compose components. Depending on their work and interests, they may master some notation (e.g., BPMN) for composition. We distinguish three sub-classes of end-user programmers:

    - Domain experts are people who compose software artifacts in the context of specific, limited domains they are familiar with. Examples of domain experts are secretaries or accountants who use spreadsheets to automate bookkeeping tasks, scientists that develop scientific workflows, for example, to analyzed human genomes, business process modelers who model processes that involve both human actors and automated computing tasks, and the like.

- App developers are people who develop client-side configurations and scripts for component-based applications, leveraging on back end-as-a-service offers. They do not have advanced programming skills, but they are able to glue together APIs and services. Examples of app developers are people who configure content management systems (e.g., WordPress) or who develop simple Web or mobile applications starting from easy-to-use programming frameworks such as Ruby on Rails or Django.
- DevOps are advanced system administrators and/or IT managers who engage in programming, so as to automate and optimize their everyday IT operations tasks. They mediate between the needs of developers (e.g., fast new software changes) and those of IT operators (e.g., stability). More and more, DevOps complements agile software development, which, for instance, asks for the development of effective command line scripts or the design of automated system migration workflows. A platform that fits the characteristics of DevOps for the composition of services is BlueMix, which enables them to build service-based cloud applications.
- End-user app remixers are people who do not have any notion of software development or composition. They are even unaware of APIs, Web services, and UI widgets (i.e., of components). Yet, they are familiar with the Web and applications in general. Thus, they think in terms of applications (the concepts of service and application are blurred) and of simple rules to integrate them. Most notably, if-this-then-that (ifttt[17]) caters for the needs of these users: it allows them to write simple rules, such as "if a new photo about me is uploaded to Instagram, add it to Dropbox". The necessary application wrappers or API/service invocations are taken over by ifttt.

3. End-Users: These users are normal users of the Web. They search for information on the web, exchange emails, use social networks or video conference applications for communicating with friends and family. They also use different applications such as Mobility services, health services etc. for completing their required tasks.

In our work, we focus on accommodating the end-users in the End-user environment where they can integrate service information without knowing the underlying techniques for Web API composition.

2.8 Summary

As we have seen in this chapter, the development of the World Wide Web was often chaotic, uncoordinated, and unpredictable. Who would have thought that many of the simple technologies created by a handful of people in the first days of the Web would survive for such a long time while industry-driven, multi-million dollar projects such as SOAP-based Web services would quickly

[17]https://ifttt.com/discover

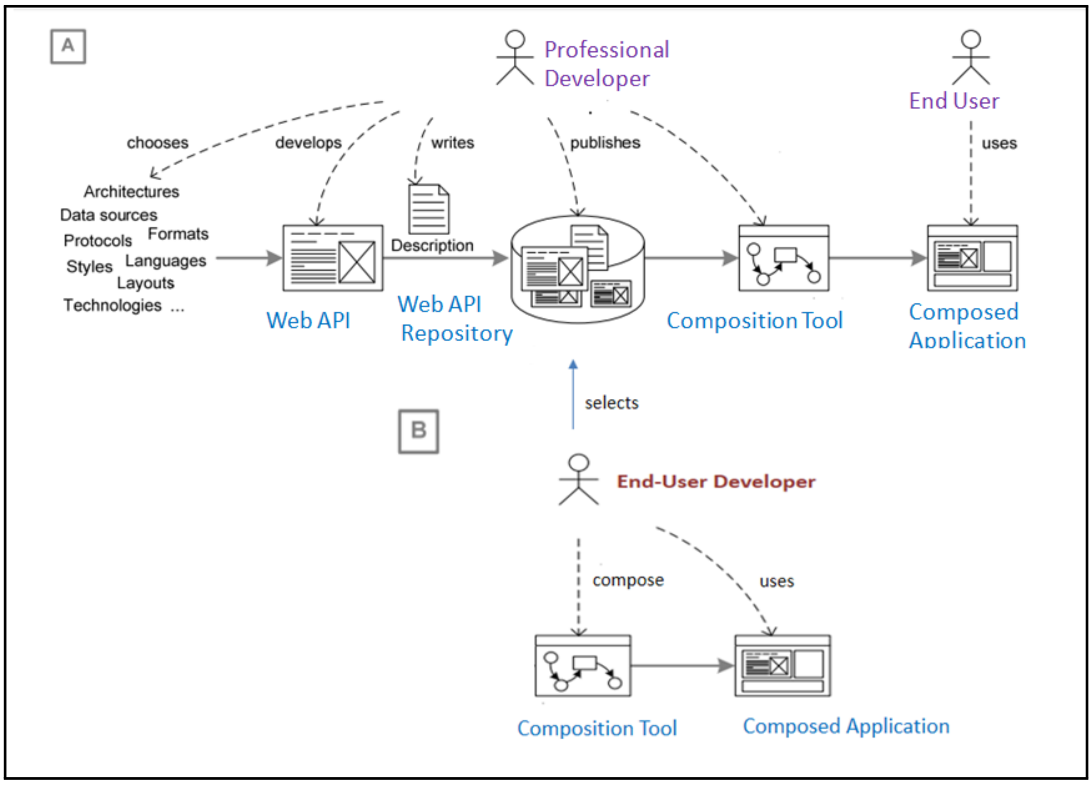


Figure 2.3: Different roles of users in the Web service composition environment (Source: [14])

pale into insignificance. Surely technical aspects played an important role but social aspects were not less important. Compared to simple JSON-based Web APIs SOAP-based services quickly "felt" heavy and complex even though, in most cases, the complexity is completely hidden by the sophisticated tooling that has been built around these technologies. The history of the Semantic Web is another case nicely illustrating this. Its acceptance languished for years but finally a simple Web page full of typographical errors formulating four very basic principles was able to herald an important turning point. The Linked Data principles did not introduce any new technology but were a mere rebranding and clarification of the vision of a Semantic Web- a Web of Data. The standardization of Web services failed, but the problems these efforts were trying to address are still valid.

The Web has grown exponentially and massive amounts of new information are being added as we speak. We have reached a point at which we humans are the bottleneck for the meaningful usage of all this knowledge. We need to extend the Web to make it easier for machines to process the available information and to exchange and manipulate data without human intervention in order to better assist us. The combination of Semantic Web technologies and RESTful Web APIs might help to bring us a step closer to this ambitious goal but, as already suggested, they suffer from various issues. Moreover, integration of services are required to be done automatically to empower end-users. In the next chapter, we will look at these issues in details and define the problems that will be discussed in this dissertation.

Chapter 3

Problem Definition

Web Services or Web APIs are becoming the glue by keeping together services within an organization while, at the same time, providing unprecedented, open access to data managed by these services to the wider world. Although several efforts have been concentrated on proper design and implementation of Web Services or Web APIs, interoperability between them remain an important issue to be resolved. Thus, automatic composition of Web Services or Web APIs become a great challenge in Service Oriented Computing arena. In this chapter, which is based on our previous works in [79–81], we will analyse the current best practices for the creation, documentation, and composition of Web services or Web APIs. We will distill a number of shortcomings and issues and finally formalize the research problems addressed by this thesis.

3.1 Interoperability Issues

Today's Web is used to share a huge amount of information through heterogeneous services or devices. Thus there is an increasing need to develop the applications termed as Web services, a set of HTTP-based interfaces to support interoperable machine-to-machine interaction by the exchange of structured data. Generally, Web services following the REST architectural style are referred to as RESTful Web services, and the programmatic interfaces of these services as REST APIs [108]. As service technologies are currently marked by the proliferation of Web APIs, RESTful services are commonly termed as Web APIs when they conform to REST principles [82, 100]. Web APIs are provided by third parties to access underlying resources, functionalities or data, through web-based user interfaces. The aim of the machine-to-machine interaction is to drive business processes in order to serve particular, application dependent goals. Due to fact that existing Web services are implemented using different protocols and are represented using different data formats, the interactions between these machine agents are resulting interoperability problems. Thus, limiting the use of Web services or Web APIs in advanced cases like automatic composition. In the following subsections 3.1.1 and 3.1.2, we discuss different practices during the implementation of Web services or Web APIs.

3.1.1 Heterogeneous Protocols

Since Web services put the emphasis on "machine-to-machine", the interactions are optimized for machines instead of being optimized for humans as websites are. In practice, two major classes of Web services can be identified, namely SOAP-based services and RESTful services as described in chapter 2. One of the major problem that led to the demise of SOAP is that the whole architecture is based on a Remote Procedure Call style which has been known to be flawed for years [140]. Furthermore, instead of using HTTP as an application protocol, it was misused as a transport protocol [99]. The complexity of the technology, which consists of far more specifications than just SOAP, WSDL and UDDI, led to a shift towards more lightweight solutions that integrate better into the architecture of the Web. This implies REST Web services, based on the REST architectural style [30]. Most services that claim to be RESTful are not as they are not fully obeying all REST constraints but primarily they use HTTP as an application protocol and that the various resources get their own IRIs. To capture the various levels of "RESTfulness", Richardson defined a maturity model [34] but by definition, a service is either RESTful by obeying to all constraints defined by REST or it is not. Frustrated by the fact that especially the hypermedia constraint is ignored by many Web services that claim to be RESTful, Fielding wrote a blog post [31] making it clear that this constraint is not optional. Since the term REST was so often misused, recently HTTP-based Web services are typically simply referred to as Web APIs instead. For services that do obey the hypermedia constraint of REST approach, the term Hypermedia API has become popular.

In order to develop a system that invokes different classes of Web services and responds proactively and intelligently, we need to provide a way that facilitate the interoperability between involved Web services or Web APIs. As a matter of facts, the HTTP protocol has been used as a universal mean for tunneling messages in business-to-business scenarios, but quite often to support additional protocols, such as WSDL/SOAP for Web Services or OGC compliant SOS/SPS for sensors, that are usually not interoperable. The more variability there is the more difficult interoperability becomes. Due to the lack of a common protocol the interoperability becomes an important research issue in the SOA community. To tackle this problem the definition of a common protocol to foster automated interoperability is needed.

3.1.2 Heterogeneous Documentation Formats

The continuous and sustained growth of ProgrammableWeb API directory[1] is only the most immediate evidence of the success that Web services and APIs have had and are having among developers [108]. Today's Web applications are leveraging available Web services or Web APIs from repositories to provide added value services. It became a common trend to use some general purpose APIs such as Google Map API, Yahoo weather API etc. to integrate in the Web Applications. Among the two core types of remote programming resources have emerged over the years: SOAP/WSDL Web services and REST APIs, the former one is having a very rich set of standards and reference specifications, and developers know well how to use WSDL [20] to describe a service and SOAP [39] to exchange messages with clients. The latter one, REST APIs leveraging REST architectural style do not have experienced this kind of standardization. While the freedom left by this choice is one of the reasons for the fast uptake of REST, it is also a reason why everybody interprets REST in an own way and follows guidelines and best practices only partially, if at all. With these varieties of interpretation of the principles and guidelines underlying REST APIs can turn into a interoperability problem. For example, when a developer has to integrate multiple APIs, if one API is provided with a suitable WADL description [44] and other API is not provided with a description at all, then it becomes a tedious task to navigate through and explore autonomously the resources managed by the API. It is even harder to create the descriptions automatically without knowing the underlying principles. If instead all Web services or APIs consistently followed the same principles and guidelines, this would result in design features (e.g., decoupling, reusability, tolerance to evolution) that would directly translate into savings in development and maintenance costs and time [94, 98]. Current uptake of REST stems the creations of different REST metadata formats that are discussed in the following section.

3.1.2.1 REST Metadata Formats

Over the last couple of years, a number of metadata standards have been emerged, mainly fueled by the need to document APIs for their consumers. As an added benefit, the same metadata is now often used to generate code (both client and server), create test harnesses, production monitors and perform real-time validation of request and response messages (when applicable). All of these provide a foundation for an improved Quality of Service (QoS) that many enterprises require as they adopt REST for their information architectures. Several frameworks have been introduced to create descriptions for REST APIs through *user-friendly* and *easy-to-use* description format editors [88].

The common goal of current metadata formats for REST APIs is to specify:

- Entry point(s)

[1] http://www.programmableweb.com/apis/directory

- Resource paths
- Methods to access these resources (GET, POST, PUT, etc.)
- Parameters that need to be supplied with these methods (Query, Template, HTTP Header, etc.)
- Formats of inbound / outbound messages/representations (JSON Schema, XML Schema, Relax NG, etc.)
- Status codes and error/fault messages
- Documentary information (descriptions, etc.) for all these.

All these are somewhat derived from the principles underlying REST APIs (i.e. those of resources, representations etc). But, Non-functional aspects of APIs, like for example authentication (Basic, OAuth, SAML, etc.), security (encryption, signatures, etc.) and versioning are unfortunately still poorly addressed by most REST metadata standards. Thus, there is a need to have a common standard to make it accessible. We listed some of the current approaches for REST metadata formats in the following:

Apiary.io: Apiary[2] has created a markdown-based metadata called the "API Blueprint" format. The parser itself is open-source and the format is well described on their website and on GitHub. Once you have an API Blueprint for your REST API, the apiary platform can be used to generate documentation, create server mocks, perform validation, etc. Although it is technically possible to use the API Blueprint format itself outside the apiary.io platform, most of the value-added functionality in the apiary.io platform seems to be proprietary.

ioDocs: ioDocs[3] is the REST metadata and documentation format of Mashery. their open-source (node.js based) platform uses JSON-formatted metadata to generate an "interactive documentation" for the described API that can be used to both learn about an API and execute ad-hoc calls against it. As opposed to the other standards described here, ioDocs also includes basic support for signatures and different versions of OAuth. As mentioned, the core ioDocs platform and tools are open-source and free to use outside the Mashery platform, but just like with apiary.io, a large number of value adding features are made available when using the Mashery platform to manage APIs.

Swagger: Swagger[4] is a REST metadata format being developed by Reverb, a spinoff from the Wordnik team. Swagger uses JSON (although it supports XML) and JSON-Schema to describe REST APIs and their parameters and messages. Just like ioDocs it includes an open-sourced UI (swagger-ui, entirely in HTML/javascript) double-serving as documentation and ad-hoc testing utility, but Swagger's strength lies in its ecosystem available at GitHub for generating both code

[2]http://apiary.io/blueprint
[3]https://github.com/mashery/iodocs
[4]https://www.swagger.io

for a number of different languages and Swagger definitions themselves via (for example) java annotations.

WADL: WADL[5] (Web Application Description Language) is an XML vocabulary that has been around for quite some time (it least in internet-time). It was submitted to the W3C by Sun in 2009 but hasn't been standardized on, perhaps due to lack of adoption. Technically it provides good support of the REST concepts outlined above; its grammar mechanism allows for the use of any standard for describing resource representations (JSON Schema, XML Schema, etc.) and there are a number of tools out there to generate code, tests and documentation from WADL specifications.

WSDL 2.0: Technically both WSDL 1.1[6] and WSDL 2.0[7] support description of REST APIs (although the support is meager in WSDL 1.1). Unfortunately though, the lack of "RESTfulness" in WSDL and its association with SOAP will probably result in its inclusion here being shrugged off as a curiosity as opposed to a valid alternative. That being said, WSDL 2.0 does technically support the description of REST APIs but its core model is not around resources/methods/representations, and the tooling is not as extensive as for the above-mentioned formats.

The REST community has been somewhat ambivalent to providing out-of-bounds metadata for REST APIs. There are Hypermedia APIs and HATEOAS ("Hypermedia as the Engine of Application State") that provide metadata about itself and the actions that are available for a requested resource as part of the response/representation itself [70]. Here, we present two of them:

HAL: HAL(Hypertext Application Language)[8] is a simplistic format available for both JSON and XML APIs to provide linking within a response. A multitude of libraries in many languages are available, both for creating and consuming HAL responses.

JSON Hyper Schema: JSON Hyper Schema[9] provides a mechanism for embedding links in JSON documents, achieving the same goals as HAL in a slightly different fashion. It is part of the JSON Schema initiative, which also includes a large number of tools for creating, parsing and validating JSON documents.

Selecting the right one from the above is not easy. The REST community should really strive to align around any one of these to avoid the fragmentation and frustration we have seen in the WS community in regard to multiple standards solving the same problem. While selecting one of these metadata formats for APIs, the following considerations should be taken:

- Is it easily accessible?

- Are the included tools available for the platform?

[5]http://www.w3.org/Submission/wadl/
[6]https://www.w3.org/TR/wsdl
[7]http://www.w3.org/TR/wsdl20/
[8]http://stateless.co/halspecification.html
[9]http://json-schema.org/

- Is there relevant tooling available given the target audience and their needs? (Testing, code-generation, etc.)
- Does it support metadata at the level needed while in use; for example the user might decide to create schemas (XML, JSON, whatever) for the resource representations, this is one area where the metadata formats differ.
- Does it support QoS metadata that user might want to make available?

Among the above mentioned data formats Swagger known as OAI specification[10], has got traction in the developer community due to its capabilities to generate great looking browser-based documentation and an ad-hoc testing interface.

3.2 End-User Development

With the technological advancement, current trend in the software supplier community is to convert their products into Web Services (an approach termed as “Software as a Service”), and all sorts of software solutions are readily available in the shape of services scattered over the Internet [21]. Most of these approaches target end-users that are generally unfamiliar with the details of the technology used to implement services. Thus, the users are facilitated to use these services to their own advantage as they used to be able to use commercial software products in the past [117]. Moreover, there are Web repositories like Programmable Web to provide compilations of available services, together with examples, guidelines and success stories in service use. Although these repositories are enabling the end-users to access these resources easily, some expertise (e.g. programming knowledge, knowledge of SOAP, WSDL, BPEL, REST etc.) are required to use these [43]. This breach between the high availability of web resources and the low prospects of their use by end-users has led many large software enterprises to create mashup development environments targeting end-users. End-users should be able to create their own software solution that exactly meets their requirements within a very short development time by composing a solution from heterogeneous resources and their front-ends [24]. There are key proposals offering DIY [78] guidance on evolving SOAs to meet end-user demands and requirements, like iGoogle[11], Yahoo! Pipes and Yahoo! Dapper[12], OpenKapow[13] or EzWeb[14]. Their aim is to get end-users to appreciate the benefits of Web services by fostering composition, loose coupling and reuse on the front-end layer, and moving towards a user-centred service as opposed to the traditional B2B conception [115]. However, existing solutions have several

[10]http://openapis.org/specification
[11]http://www.google.com/ig
[12]http://pipes.yahoo.com/
[13]http://kapowsoftware.com
[14]http://http://conwet.fi.upm.es/morfeo-project/ezweb blog/?lng=en

weaknesses: one of them is that they fail to provide a user-based front-end to enable the user-centered composition of back-end services [76]. Solutions tend to be limited to the combination of just content rather than applications, overlooking end-users and their participation. The major problem with these tools is that end-user developers are often unable to translate their particular requirements into a specific software product [65, 75], because each tool focuses on achieving a particular solution type that does not necessarily meet user needs. For example, Yahoo! Pipes creates a correctly filtered data list feed, Kapow Platform creates an execution flow based on pre-existing interlinked web portals, and so on. Users who require a more complex need to solve a problem other than for which the tool was designed will be disappointed. Moreover, these tools do not match the way in which end-user developers conceive their solution; nor do they offer a natural development process for end-user characteristics and needs. Some approaches have been developed to consider these limitations such as [76, 77], address these issues and proposed a visual mashup composition framework using component- and connector-based approach for end-user composite web applications development. Also there are some approaches like [4], [84] etc. are providing support for end-user development that requires little programming knowledge. Additionally, supports for end-users without any programming language have been supported by different approaches(e.g. [37]) or tools like IFTTT[15], Zapier[16] etc. These tools are facilitating the end-users to develop a composite applications in a drag-and-drop or trigger-action manner, but not considering the capabilities of all kind of user groups. At best of our knowledge, there is to date, however, no approach has considered the elders as end-user developers in Software-oriented architecture by analyzing their requirements and barriers.

3.3 Summary

The Web has grown exponentially and massive amounts of new information are being added as we speak. We have reached a point at which we humans are the bottleneck for the meaningful usage of all this knowledge. We need to extend the Web to make it easier for machines to process the available information and to exchange and manipulate data without human intervention in order to better assist us. The combination of Semantic Web technologies and RESTful Web APIs might help to bring us a step closer to this ambitious goal. Needless to say that data integration is also made much more difficult given that data models differ widely and all the semantics are implicit. If the semantics were explicit and the data models were generic, data integration would be drastically simplified. In fact, data could be integrated (semi) automatically with other data sources. For instance, a typical mashup combining and showing data from different sources on a map could be created automatically. The widget would be able to automatically figure out which parts of the representation represent the needed coordinates and in consequence render the data on the map.

[15]https://ifttt.com/discover
[16]https://zapier.com/

Today, more and more services published on the Web are claiming to be designed using REST, but actually missing some of the features, like the hypermedia control, which is crucial to automate self-management operations. The use of hypermedia as the engine of application state [30] is a central aspect of the REST architectural style and when building traditional Web sites, developers intuitively use it to guide visitors through their sites. They understand that no visitor is interested in reading documentation that tells them how to handcraft the URLs necessary to access the desired pages. Developers spend considerable time to ensure that their sites are fully interlinked so that visitors are able to reach every single page in just a few clicks. To achieve that, links have to be labeled so that users are able to select the link bringing them one step closer to their goal. Often that means multiple links with different labels but the same target are presented to make sure that a visitor finds the right path. This is most evident when looking at the checkout process of e-commerce sites which usually consists of a single path leading straight to the order confirmation page (plus a typically de-emphasized link back to the homepage or shopping cart). These practices build the foundation of today's Web, a gigantic graph consisting of billions and billions of interlinked pages. Hyperlinks are such a fundamental building block of the Web architecture that it feels natural to browse across sites from completely different publishers. It is taken for granted that content links to other relevant content; relevant links are generally seen as a sign of quality. Surprisingly, Web services very rarely link to external data. As a matter of fact, most times even links to other resources within the service itself are missing. The problem with the practices outlined in this chapter is that Web services or Web APIs are implemented using different protocols, which have different data formats that are not operable. The interaction between Web Services or Web APIs that generally seen as a data integration problem, is an important issue to be resolved when we are considering composition scenarios. To accomplish this, correct and complete use of the HTTP protocol to publish, manage, and operate services on the Web by fully exploiting the REST principles is needed. Another problem described in section 3.1.2 is that Service providers are providing descriptions in different formats or even not any description at all. As described in subsection 3.1.2.1, several description tools and formats have been introduced for describing REST Web APIs both in human and machine readable formats. Although these descriptions provide functional information about the APIs (e.g. HTTP methods, URIs, model schema, etc.), the information that qualifies the properties of APIs (e.g. classification of input arguments and response data) is missing. To fulfill users' specific needs there is a need to provide a complete set of information to the users that will facilitate the composition of APIs. Other aspects, such as documentation of these Web Services or Web APIs needs to be done in a similar way and appropriate tools are needed to be provided. We believe that it should be feasible to standardize and streamline the development of Web services or Web APIs by combining ideas and principles from both the REST architectural style and the Semantic Web.

In contrast to the current practices for Web APIs, Linked Data is useful for describing data in a machine-readable way. All data publishers have to do, is reuse one or more of the many existing vocabularies to express their data. This not only eliminates the need to document the semantics of the data over and over again but also improves the interoperability of systems exchanging such data and the reusability of code. It would thus make sense to reuse these technologies and vocabularies for the creation and description of Web APIs. All a developer has to do is annotate the code with concepts from a shared vocabulary. The positive side effect of such an approach is that machines would be able to recognize when equivalent elements are encountered and process them using existing code instead of requiring manual adaptations by the implementer. In fact, the documentation could be linked directly to the messages which would make them self-descriptive and thus reduce the need for additional human-readable documentation. Although, these solutions presents API description with semantic annotations, most of them fails to add semantic annotations automatically or semi-automatically [137].

Beside these technical issues, a lot of developers are also simply overwhelmed by the complexity, perceived or otherwise, or are just reluctant to use new technologies [40]. The prevalent terminology, suffused with words such as Ontology, just seems to fuel their misconceptions. Furthermore, the fact that the Semantic Web community derailed into the artificial intelligence domain instead of concentrating on more practical data oriented applications certainly played a major role in that regard as well. A lot of potential users were alienated by this and developed an aversion to Semantic Web technologies, a phenomenon we denoted as Semaphobia [4]. Moreover, advanced technical skills are needed to manage semantics and compose different Web APIs, which reduce the number of potential end-users of such solutions. A solution to this problem might be a more gradual introduction to those principles and practices by the use of less disruptive technologies. Clear incentives along with simple specifications and guidelines are necessary. Also, understanding end-users' characteristics, requirements, barriers and attitudes need to be analyzed to provide a way to include all groups of end-users in the end-user development paradigm.

Having identified several issues and shortcomings of current best practices, we are able to formalize the problem statements that integrates the above discussed issues into the primary aim of this thesis, i.e., to support end-users to compose Web APIs. The problem addressed in this thesis consists of three clearly defined sub problems that are summarized as follows:

1. Interoperability between Web services or Web APIs is not supported due to the implementation of Web services or Web APIs using different protocols or architectures.

2. Heterogeneous description formats are available that only provide functional information about the APIs (e.g. HTTP methods, URIs, model schema, etc.) but the information that qualifies the properties of APIs (e.g.classification of input arguments and response data) are missing.

3. There is a mismatch between the technical functionalities and visual elements provided by the Web service or Web API composition tools and the end-users' capabilities to create a

system that is not only operational but also usable and adaptable to all kind of user groups.

We believe these three issues need to be resolved to provide an appropriate solution for user-driven composition of Web APIs. Our next chapter will discuss the related works.

Chapter 4

Related Work

Since the introduction of Service Oriented Architecture (SOA), it has got the prevalence in the deployment of enterprise applications and integration of these applications within and across organizational boundaries. Public repositories (e.g. ProgrammableWeb[1]) publish a number of various services in the form of API provided by anonymous developers and any integrator can use them in their own applications via little implementation efforts. Moreover, several tools have been developed that persuaded many people to integrate services despite having little programming knowledge. Although, a simple task can be accomplished by invoking a service like weather information, location maps or payment, a complex task often requires numerous interactions among services that collectively complete the task. To facilitate an automatic composition process that can provide a composite service with required functionalities, all the component services must be interoperable. A lot of research has been done to solve different issues regarding automatic Web service composition following different approaches. In this chapter, we discuss some of the approaches that address the Web service or Web API composition and End-user development issues. To analyze the current state of the art aiming to facilitate user-driven API composition, we discuss existing approaches that facilitate the use of Web APIs by machine agents. We also analyze the existing approaches and tools considering API Descriptions, Semantic Annotation and Composition.

4.1 Web Service Composition

Web service technology has emerged as the most promising choice to implement SOA and its strategic objectives. A Web service is essentially a semantically well-defined abstraction of a

[1]www.programmableweb.com

set of computational or physical activities involving a number of resources, intended to fulfill a customer need or a business requirement [120]. A Web service could be described, advertised and discovered using standard-based languages, and interacted through Internet-based protocols. Two types of Web services are most popular and widely used: SOAP-based Web services and RESTful Web services. The former is based on Web Services Description Language (WSDL)[2] and Simple Object Access Protocol (SOAP)[3] while the latter conforms to the REST architectural principles [30]. Since the introduction of REST, it has been emerged as the choice for many of the leading Internet companies to expose their internal data and functionalities as URI identified resources [152]. In REST system, servers and clients typically travel through different states of resource representations by following the interlinks between resources. In contrast to the operation-centric perspective of WSDL/SOAP Web services, RESTful Web services view the applications from a resource-centric perspective. Due to these differences along with the differences in the adopted styles (imperative against declarative), the automated composition issues are different for these two kinds of Web services. In the past decade, much research efforts have been put on emphasizing automated approaches to WSDL/SOAP and RESTful Web service composition [106, 111, 120]. WSDL/SOAP Web service composition approaches focus on functional composition of individual Web services. That is, how to compose a new functionality out of existing component functionalities. However, RESTful Web Services model the system from the perspective of resources. The composition of RESTful Web Services focuses on the resource composition and "state transfer" between candidate Web Services. Services interact by exchanging request and response messages, each of which includes enough information to describe how to process the message. In contrast to SOAP-based Web services, RESTful Web services are lightweight and stateless, which are well suited for tactical, ad hoc integration over the Web. Several issues have been identified regarding Web service composition from the above mentioned survey papers [106, 111, 120]. One of the major issue is to handle the composition of heterogeneous Web services that lacks interoperability due to different protocols and data formats they use.

4.1.1 Interactions between Heterogeneous Web Services

Nowadays, variety of services on the Web are increasing considering pervasive computing environments where different devices often use different technologies, which make the communication complex and sometimes even troublesome. Although, the HTTP protocol has been used as a universal mean for tunneling messages in business-to-business scenarios, interoperability between protocols such as WSDL/SOAP for Web services or OGC compliant SOS/SPS for sensors is missing. We consider this issue and analyze different approaches presented in the literature that discuss the adoption of REST architectural style in the Sensor Web.

[2]http://www.w3.org/TR/wsdl
[3]http://www.w3.org/TR/SOAP

In the domain of SOAP/WSDL services, messages are exchanged between endpoints of published applications by using the Web as a universal transport medium. In this way, the applications interact through the Web but remain outside of the Web. In addition, SOAP is the single standardized message format in this approach and messages are exchanged in both directions by using only one HTTP verb (POST). There are several papers that compare SOAP and REST such as [99]. The major advantage of adopting full semantics of HTTP verbs to expose operations is that applications become part of the Web, making it a universal medium for publishing globally accessible information. While sensors are considered as services, the adoption of the REST approach in sensor services is needed. In the domain of sensors, the most relevant works are related to OGC standards. In [58], the authors have given a Linked Data model for sensor data and have implemented a RESTful proxy for SOS in order to improve integration and inter-linkage of observation data for the Digital Earth. We have been inspired from their work regarding the URI scheme for observations data, and the creation of meaningful links between data sets. According to the view presented in [145], the sensor data can be properly linked following the REST architectural style. Having sensor data properly structured on the Web is fundamental whenever integration needs to take place as in sensor networks. Further approaches have addressed the discovery, selection and use of sensors as a service. An advanced approach has been presented in [41] that offers search mechanisms of sensors that exploit semantic relationships and harvest sensor metadata into existing catalogues. In paper [59], a model-based, service-oriented architecture has been used for composing complex services from elementary ones, on sensor nodes. The selection process evaluates the processing and communication cost of the service. In [9], the authors focus on service composition in pervasive systems. They propose ranking services based on context-related criteria so that the selection is based on the service matching score with the composition features. All the above mechanisms need to address critical aspects like the heterogeneity of interfaces and data models mismatches, and thus, can be used for the integration of sensor data in sensor networks. In paper [95], the authors describe the selection of services that match user preferences by collecting and evaluating services' descriptions. RESTful interactions can be integrated into such mechanism to facilitate the descriptions discovery and the services selection, and to enable pervasive systems make use of the selection process. Such solution minimizes the number of services and avoids unsuitable services in pervasive systems since it involves only the services that meet the user requirements. The approach described in [26], investigates the use of the REST architectural style for providing the functionality of sensors in pervasive systems. It emphasizes the abstraction of data and services as resources, services interoperation via self-describing data and services orchestration with loosely typed components. In [109], the DIGIHOME platform has been developed to deal with the heterogeneity, mobility and adaptation issues in smart homes where devices have advanced computational capabilities to improve the user satisfaction, but the heterogeneity of protocols used constrains the integration of these devices into a larger monitoring system. The platform provides software connectors for devices accessed by a variant of protocols such as ZigBee, SOAP and CAN, while HTTP is the communication protocol for the detection of adaptation situations and the handling of events. In [41], the authors explore REST as a mean to build a "universal" API for web-enabled smart things. They give emphasis on the decoupling of services from their representation and the negotiation mechanisms for the representation format, and they propose AtomPub to enable push interactions with sensors, and gateways that abstract communication with non Web-enabled devices behind a RESTful API. Although the approaches presented in [41, 109] use HTTP according to the REST principles, they do not make explicit how services with conventional interfaces are mapped to a RESTful API.

4.2 Web Service or Web API Descriptions

SOA standards such as WSDL, SOAP and further WS specifications were devised in order to provide the necessary technologies to support the development of software applications by reusing remote services, that is software components offered via programming language independent interfaces [100]. Due to the absence of explicit semantics of these services and the data they manipulate, WSDL requires substantial manual effort to locate,interpret and integrate existing services. Several approaches have been proposed such as OWL-S[4], WSMO[5], and SAWSDL[6] to overcome these limitations by enriching Web service descriptions and data models with semantic annotations. However, these approaches remain incompatible due to the use of different representation languages as well as because of certain conceptual differences. Despite of having these differences at the semantic level, most of the initiatives taken by the Semantic Web Service community were based upon the semantic enrichment of WSDL Web services, which have turned out not to be prevalent on the Web.

The proliferation of Web API has been marked in the world of services on the Web, due to the flexibility and simplicity they provide to access the data and functionalities of the services. By observing the directories like ProgrammableWeb[7], it can be stated that most of the service providers are making their services accessible through APIs and thus, facilitating third parties to combine diverse APIs into Mashups to provide added value solutions. These APIs are usually called as RESTful services [83, 107] when they conform to REST principles [30]. Although the users can search for APIs in repositories, well-structured API documentations enabling effective discovery are missing. Several frameworks have been introduced to create descriptions for REST APIs through *user-friendly* and *easy-to-use* description format editors [88]. Some REST metadata formats have been created to document REST APIs in a consistent way. These standards offer a way to represent an API by specifying entry point(s), resource paths, methods to access

[4]https://www.w3.org/Submission/OWL-S/
[5]https://www.w3.org/Submission/WSMO/
[6]https://www.w3.org/TR/sawsdl/
[7]http://www.programmableweb.com

these resources, parameters to be supplied with these methods, formats of inbound/outbound representations, status codes, error messages and documentary information. Some of the most popular standards are the following: Swagger[8] or Open API Initiative specification, which offers a large ecosystem of API tooling, has a very large community of active users and great support in almost every modern programming languages and allows developers to test the APIs immediately through easy deployment of server instances. API Blueprints, where an API description can be used in the Apiary[9] platform to create automated mock servers, validators etc. The Hydra [70] specification, enrich web APIs with tools and techniques from the semantic web area. RAML[10] is a well-structured and modern API modeling language. Swagger is obviously the dominant choice at the moment, though all specifications are promising [132]. This statement is convincing because of its above mentioned promising features and also because it has the capability to provide human-readable API descriptions by using YAML, as well as JSON. Moreover, it defines a standard in the Web API description method, being partner of OAI specification as opposed to other specifications. We believe that the objective of Open API Initiative to create an open description format for API services that is vendor-neutral, portable and open will accelerate the vision of a truly connected world. Although API Description created by Swagger and other editors give the opportunity to create descriptions easily, they lack detailed information qualifying the properties of an API (e.g. classification of input arguments and response data), which is relevant to address automatic discovery and composition performed by machines. To address these issues, additional information to are needed to be added in the descriptions.

4.2.1 Existing Approaches and Tools for Semantic Annotations to Facilitate Composition

Extensive research has been conducted with the vision to create automatic integration of Web Services or APIs [120]. But in practice most of these approaches are having problems in communicating between candidate Web services or APIs due to the lack of semantic correlation of properties. To automate the interactions between Web services or Web APIs there is a need to describe the exchanged data with semantics. To achieve this there are two possibilities, one is by directly creating Web service descriptions following specifications defined in a logic based language, like the Web Ontology Language (OWL) and the second one is by linking existing descriptions to these ontologies (i.e. aligned descriptions to shared domain ontologies). As the first approach needs expertise in logic based languages, its adoption is curtailed. The latter is more approachable because it enriches the existing descriptions to be remain compliant with other semantic descriptions. Thus, this approach reduces the possibilities to lock out non-semantic

[8]http://www.swagger.io

[9]https://apiary.io/

[10]http://raml.org/

descriptions.

To support automatic composition of Web Services, several approaches have been proposed focusing on semantic annotations, but many of them are either not validated or the validation lacks credibility [131]. To accelerate the use of Web APIs by machine agents, Semantic-Web researchers proposed a number of solutions. Paper [137] emphasises on the need to provide self-descriptive descriptions that include metadata, that can be interpreted by machine agents in a bottom up way (i.e. information structure should be in pieces to whole). Paper [96] proposes a set of best practices to build self-descriptive RESTful services accessible by both humans and machines. Moreover, it defines a framework that extracts compliant descriptions from documents published on the Web, and makes them available to clients as resources. Paper [70] develops Hydra, a small vocabulary to describe Web APIs; this approach aims to introduce a new breed of interoperable Web APIs by breaking the descriptions down into small independent fragments. Paper [86] focuses on facilitating composition process by reasoning with tailored ontologies that capture user preferences.

To analyse current approaches regarding Semantic Descriptions of Web APIs, our discussion includes WSDL, WADL, hREST, RDFa, MicroWSMO, SA-REST, MSM, RESTdesc, SEREDASj following the discussions in [69, 121, 136, 139]. WADL is specifically used for syntactic descriptions of RESTful services, instead of WSDL, which is used to describe Web Services in general. Both of them, however, do not support simple links and they appear to be too heavy to describe Web APIs. hREST and SA-REST are more approachable as they use microformats which are embedded in the Web page of the API documentations. Although these two approaches are more useful for the semantic correlation, they are not focusing enough on the description itself. RDFa follows the same consideration made for hREST and SA-REST about the specialisation in doing semantic-annotation, turning out to be even more complex to use [136].

For the purpose of our work we place great emphasis on the analysis of previously listed approaches specific to Web APIs. MicroWSMO relies on hREST offering service property descriptors, but it also focuses on the semantic part of the descriptions. RESTdesc is a logic-based Web API description method that captures the functionality of Web APIs, describing an HTTP request and its preconditions and postconditions expressed in Notation3 (N3), which is a serialization form for the main Semantic Web language, RDF [139]. However, RESTDesc requires manual effort to produce the desired specifications [136] and also there are some complex use cases that cannot be covered, such as cases in smart environments where RDF or N3 are not providing proper solutions. SEREDASj provides a way to describe Web APIs with JSON-based method that is simpler to apply. However, it produces two different documents in order to provide a complete descriptions, proving to be difficult to maintain.

There are also different tools using semantic annotation for composition, such as OntoMat [45] and METEOR-S [97], Kim [103] and AeroDAML [66], etc. For the purpose of our work, we analyse two tools SWEET[11] and Karma[12], which emphasise on user driven integration of Web APIs by enabling semantic annotations.

SWEET is a tool that allows the development of mashup based on linked open data and services, by enabling the creation of semantic descriptions of Web APIs. The input is the HTML Web page describing a Web API and the result is a semantically annotated HTML page, or a RDF MicroWSMO description. To obtain this result, users need to identify the properties of the service by inputting hRESTS tags into HTML service descriptions. After this step, users define appropriate domain ontologies to annotate the service properties with semantic information. Users can then save or export the annotated Web API.Although SWEET allows the definition of semi-automatic annotations, the users have to make long effort because they need to find all the parameters to be annotated in Web APIs description pages.

Karma [42, 129] is another tool which allows users to integrate data from different data sources, including databases, spreadsheets, XML, JSON and Web APIs. The inputs to the process are an ontology, a data source and a database of semantic type that the system has learned to recognise, based on prior use of the tool. The system is based on a probabilistic model that is also capable of learning, with a model named conditional random field (CRF) [68], whenever users define a new mapping from data source in the ontology. The process consists of four steps (i) Assign Semantic Type, (ii) Construct Graph, (iii) Generate Source Description and (iv) Generate RDF.

Both of these tools guide users in the process of composition of Web APIs and endeavour to suggest the correct annotations, based on the use of ontologies. The main difference between SWEET and Karma is that SWEET allows the addition of hREST tags in the HTML page, since it uses only HTML pages as inputs. Karma, instead, employs a table annotation technique creating a table where, once properties are input into the header row, API responses are populated in the columns. These properties are collected dynamically through different invocations of an API, by defining several different parameters to retrieve the most accurate representation. However, this tool does not consider the context outside tables.

Paper [150] use a different technique named as “TableMiner”. TableMiner is an innovative approach to classify table columns and disambiguate cell contents following different algorithms. This approach considers two types of contexts, one is defined as “in-table context”, including column header, column content and row content, and another one is “out-table context”, which could include semantic mark-up already inserted in a web page, the web-page title, paragraphs and table captions. The usage of this out-table context and the previously mentioned algorithms are taking TableMiner a step forward in the State-of-the-Art. We are convinced with this approach.

[11]http://sweet.kmi.open.ac.uk/index.html
[12]http://usc-isi-i2.github.io/karma/

4.3 End-User Development Approaches

Technical evolution and proliferation of new devices such as mobile phones, net books, tablets, eBook readers, but also radios, TVs and other multimedia devices, has however brought a high level of diversity with respect to the means used by people to use the internet. Currently, all sorts of devices are used on a daily basis to access or provide information as well as to exploit functionality exposed on the Internet (including the Web). In one hand, the pervasiveness of these devices provides an unprecedented level of accessibility to the Internet, allowing users to use their favorite resources and services at anytime anywhere. On the other hand, Service-oriented architectures (SOAs) enables end-users to create their own software solutions by composing and orchestrating heterogeneous internet services, enabling real end-user development [73]. Existing software development tools dedicated for end-user developers work well for small problems, but do not produce valid solutions for more complex problems [77]. There are many initiatives taken by Software supplier companies like Microsoft, Apple, IBM, Yahoo! etc. to support end-user developers by providing tools such as iGoogle[13], Yahoo! Pipes and Yahoo! Dapper [14], OpenKapow[15] etc. Their aim is to get end-users to appreciate the benefits of web services by fostering composition, loose coupling and reuse on the front-end layer, and moving towards a user-centered service as opposed to the traditional B2B conception. Although these tools offer do-it-yourself guidance on how to evolve end-user developments to meet end-user requirements, support for more complex applications are missing [110]. Another similar approach is a component- and connector-based visual composition framework named as FAST [76] that enables non-programmer end-users to develop composite web applications.

There are many approaches that show the prevalence of end-user development in Web environment such as [14, 37]. Main features of these approaches describes the deployment types of the tools, required level of programming knowledge to properly use the tool, reuse of existing web resources such as content, functionality etc., component selection processes (e.g. by direct selection or via custom scripts) and the process of component composition among selected components. Table 4.1 describes the Web tools (following [37]) where the end-users should have little programming knowledge. These tools are deployed as applications. Table 4.2 describes the Web tools where the end-users don't need to have programming knowledge and can be accessible through web browsers.

[13]http://www.google.com/ig
[14]http://pipes.yahoo.com/
[15]http://kapowsoftware.com

Table 4.1: EUD Web tools for end-users with programming knowledge

Reference	Reuse of Existing Sources	Component Selection	Component Composition
d.mix [46]	Components visible on the browser to obtain new applications	Direct picking from Web UIs	Editing of Existing HTMLcode, script insertion.
NaturalMash [3]	Automation of access to existing Web services mapped into “ingredients”.	Development of “ingredients” from existing Web services	Natural language and drag-and drop metaphor.
PEUDOM [84]	Automation of access to existing Web services mapped into PEUDOM components.	From PEUDOM Component Editor	Drag-and-drop metaphor and other settings.

Table 4.2: EUD Web tools for end-users without programming knowledge

Reference	Reuse of Existing Sources	Component Selection	Component Composition
Sifter [55]	Components visible on the browser	Composed to enhance the application	By direct picking from Web UIs , Full automatic or user-adjusted
Mashup Editor [37]	Components visible on the browser to obtain new applications	Direct selection from Web UIs	Copy and paste metaphor between existing Web components
IFTTT[16]	Components visible on the browser to obtain new applications (e.g Service or Applets)	Direct selection from existing Web Services	Trigger and Action settings to create “Applets”
Zapier[17]	Components visible on the browsers to obtain new applications (e.g. Zap)	Direct selection from existing Web Applications	Trigger and Action settings to create “Zap”

4.4 Summary

Over the last decade, Web service composition has become a thriving area of research and academic efforts, with a substantial body of research work produced. In this chapter we discuss some of the existing Web service or Web API composition approaches along with semantic annotation and end-user development approaches. We analyzed related issues needed to be resolved to facilitate semi-automatic composition driven by users. To facilitate automated composition, existing literature shows the need to have a common way for communication among the involved heterogeneous Web services or Web APIs. As REST is the most dominant approach, we emphasize on the RESTful interactions considering pervasive computing environment. There are existing machine-understandable description formats for RESTful APIs. Existing literature emphasize on automating the process of composing RESTful Web APIs, from a pool of vast amount of candidate RESTful Web APIs. To do that there should be an automatic way for adding semantic annotations in the Web API descriptions. Moreover, the gap between the users' requirements and Web API composition should be taken into consideration to empower all groups of end-users in the end-user development paradigm.

Chapter 5

Set the Context: End-users' Requirements

The twenty-first century phenomenon is known as the "demographic time bomb", as we experience an increase in the number of elderly people and a reduction of younger people to care for them as they lose their independence in later years [147]. This trend has prompted many researchers and developers of information technology to find ways to enable elderly people to live independently for longer. Although a number of research projects have sought to develop smart home environments to meet the special needs of the aging population and offer them security, comfort and quality of life, initiative to make them independent in information seeking and utilizing these information to complete their daily needs is missing . Thus, we emphasis on this specific group of end-users and analyze the context of elder domain.

5.1 Context

According to the statistics presented in the literature [28, 133], the age structure of the world population is projected to be dramatically changed in the coming decades due to the dynamics of fertility, life expectancy and migration rates. The EU population is expected to be increased by almost 4 percent (from 507 million in 2013 up to 2050), when it will peak (at 526 million) and will thereafter decline slowly (to 523 million in 2060). In the United States, projected population of aged 65 and older is 98.2 million in 2016. The projected estimates also show that in 2033, for the first time, the population of 65 and older would outnumber the people younger than 18 in the US. These studies also show that 71 percent of them use computers at home and additionally, 62.4 percent accessed the Internet through a high-speed Internet connection. To face the challenge of this aging structure the countries of Europe and US respond with the solution in investing in people to allow them to age actively. Active aging will aim at enhancing the motivation and opportunities for people to participate actively throughout their lives. While governments and EU and US are concentrating on fostering higher labor force participation across all age groups and health care for elders, the active participation in the society and fulfill their own needs with new technologies is a challenge of this aging phenomenon. Higher participation in using new technologies will also ensure increase in acquiring skills. These efforts include interests in new technologies and continued with training and updating of knowledge and skills throughout the life cycle.

Since its inception, the Web has been a medium that constantly surpassing itself. With the increasing availability of high-speed, broadband access, for many the Internet has become a lifestyle. We have seen the integration of audio, video, plug-ins and innovative designs that have revolutionized the way we think about and interact with the Web. Moreover, applications have been designed considering elders' various needs. Yet, the design, content and functionalities of many of these applications typically reflect the personality and interests of the designers and developers who typically are one-third the age of the senior audience. Successful development will be those who understand the realities of human factors for young and old alike. Understanding these realities of human factors allow us to realize the potential of the personal devices such as personal computer, laptop, mobile, tablet in daily activities. Moreover, the Internet becoming the most liberating technologies since the invention of the fountain pen, allowing people of all ages to enhance their community, independent living, creativity and employability.

The first step to design for different age segments and demographics is to understand generational perspectives. From a user perspective, it is important to think about the following aspects:

- how the end-users (including elders) use the Web?
- Which kind of tasks are important to them?
- What devices(s) usually they use?
- How will they be connected (dial up or broadband)?
- What environment will they be using it (work, home and/or travel)?
- What limitations (physical or cognitive) might they have?

We base our research on the premise that the revolution of web technology enables end-users to search for information according to their requirements, resulting older adults to maintain their independence. A coherent suite of technologies will eventually let older adults to be proactive about their own health care, will aid them in daily activities and help them to learn new skills, will create new avenues for social communication, and will help ensure their safety and well being.

In our work, we also take the consideration that today's elder people are using technologies for different purposes. They are able to use new systems, applications or tools if those are provided with a easily accessible and simple to use platform. An issue here is what might appear as natural and intuitive to developers and designers may be confusing to the end-users [4] and we believe, that is also true for elder users. To resolve this issue, we analyse the requirements of a specific group of users (i.e. elders) to provide special supports they need while using the systems or tools. We also consider the effective interface usability that takes into account navigability, intuitiveness, functionality and interactivity from a user perspective, rather than the perceptions of the designer, developer or organization.

5.2 Understanding Elders' Needs and Attitude

Although the internet has grown tremendously and almost invaded every aspect of our lives, several barriers still exist for older people to really make it an irreplaceable part of their daily lives due to a rapid decline in fluid intelligence (processing speed, cognitive flexibility or ability to switch processing strategies, attention control and visualization span) as well as in motor skills [54, 123, 141]. Some of these abilities are directly linked to the skills required to process information from the Web. Because of this, old people are known to be slow and less efficient when using the Web. They click on less umber of pages, take longer to click, make unnecessary repeated visits to already visited pages, etc.[17, 93]. However crystallized intelligence (prior knowledge, experience and vocabulary skills) increases and/or becomes stable with aging. Therefore it is higher for older people compared to younger. Despite having physical challenges and difficulties in learning new technologies elderly are proponents of technological advancement, which may provide valuable opportunities for them. To understand their requirements toward composition environment, we have divided the requirements into five parts: *requirements for age related changes, social requirements of elders, elders' behavior towards the system, perception of the use of internet, web and web applications by elders and role of elders in searching information on the Web* .

5.2.1 Requirements for Age Related Changes

There are different barriers and challenges that elders are having to adopt new technologies [146]:

- Physical challenges: Around two in five seniors indicate that they have a " physical or health condition that makes reading difficult or challenging" or a " disability, handicap, or chronic disease that prevents them from fully participating in many common daily activities".
- Skeptical attitudes about the benefits of technology: Half of the non-users (49 percent) agreed with the statement that "people lacking internet access are at a real disadvantage

because of all the information they might be missing", with 25 percent agreeing strongly. But 35 percent of these older non-internet users disagreed that they are missing out on important information and 18 percent of them strongly disagreed.

- Difficulties learning to use new technologies: Just 18 percent would feel comfortable learning to use a new technology device such as a smart phone or tablet on their own, while 77 percent indicated that they would need someone to help walk them through the process.

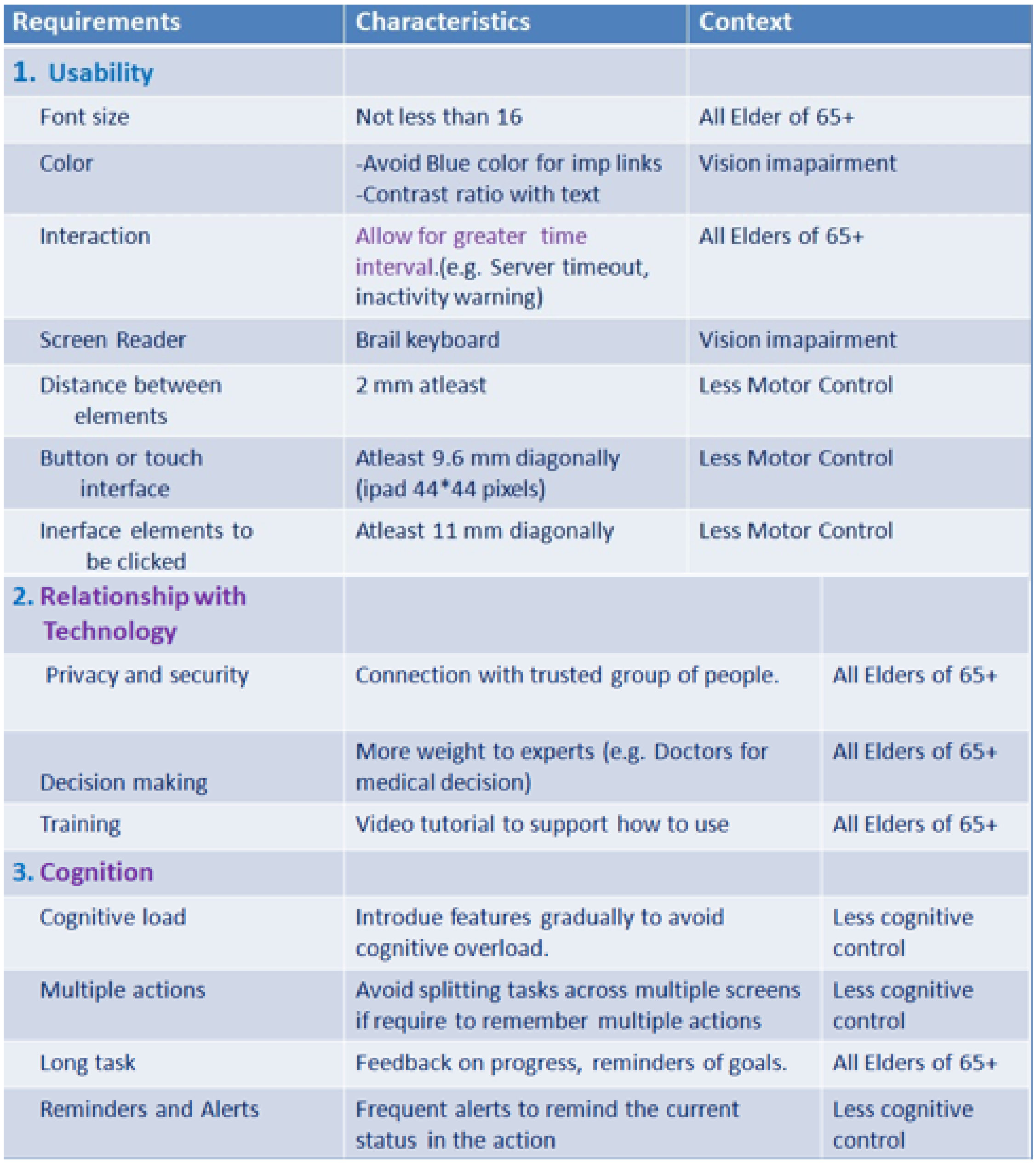
Moreover the perception of a lack of benefits and irrespective of perceived costs are reason enough to reject a new technology. In general, elderly are proponents of technological advancement, which may provide valuable opportunities for them, but not at any price. They do not want technologies that replace face-to-face contacts, but are interested in technology that supports additional social contacts, connecting people with similar interests (e.g., clubs), or helping them to stay in touch when immobile. Along with physical barriers, age related memory changes and their effects on learning are no doubt at the heart of the difficulties which older people have in using new technologies [148]. Although old age is associated with a decline in intellectual skills which affects the absorption of new information, some skills are not affected by aging and research has shown that certain types of memory are unaffected by aging [123]. According to [13] and [64], design principle should consider the following aspects to meet elders' needs:

- vision and hearing of the elderly people (e.g., appropriate font size, usually bigger than 16 pixels, contrast ratios with text, or provision of subtitles when video or audio content is fundamental to the user experience);
- motor control (e.g., bigger buttons, at least 9.6 mm or bigger screen device);
- device use (user-friendly);
- relationships (e.g., enable connection with a smaller, but a more important group of people such as their family members and friends);
- cognitive capacities of the elderly (e.g., provision of services such as reminders and alerts as cues for habitual actions).

Although Service Providers are concerned about the physical decline of the aging phenomena but they are missing some features like security and privacy, experience with technology, trust, issues of cognition, decision making etc. In Fig. 5.1, we summarize different requirements of elders considering three aspects: Usability, Relationship with technology and Cognition.

5.2.2 Social Requirements of Elders: Fostering Social Inclusion

To understand the social requirements of elders we have adopted the scenario-based methods [27] which is helpful for discussing and analyzing how the composition platform could be used to



Requirements	Characteristics	Context
1. Usability		
Font size	Not less than 16	All Elder of 65+
Color	-Avoid Blue color for imp links -Contrast ratio with text	Vision imapairment
Interaction	Allow for greater time interval.(e.g. Server timeout, inactivity warning)	All Elders of 65+
Screen Reader	Brail keyboard	Vision imapairment
Distance between elements	2 mm atleast	Less Motor Control
Button or touch interface	Atleast 9.6 mm diagonally (ipad 44*44 pixels)	Less Motor Control
Inerface elements to be clicked	Atleast 11 mm diagonally	Less Motor Control
2. Relationship with Technology		
Privacy and security	Connection with trusted group of people.	All Elders of 65+
Decision making	More weight to experts (e.g. Doctors for medical decision)	All Elders of 65+
Training	Video tutorial to support how to use	All Elders of 65+
3. Cognition		
Cognitive load	Introdue features gradually to avoid cognitive overload.	Less cognitive control
Multiple actions	Avoid splitting tasks across multiple screens if require to remember multiple actions	Less cognitive control
Long task	Feedback on progress, reminders of goals.	All Elders of 65+
Reminders and Alerts	Frequent alerts to remind the current status in the action	Less cognitive control

Figure 5.1: Requirements of elders considering interactions with the systems.

reshape their activities. This approach helps to describe the scenarios before a system is built and its impacts felt. We divide the application needs of elders into two categories:

- Needs for assistance
- Needs for impairment

The applications needs for assistance and impairment should consider the easy to use and high usability features following web accessibility guideline for older people[1]. Moreover we should consider the attitude towards the system along with all the physical barriers and cognitive load capabilities.

Use Case Scenario for Impairment: Mr. Rossi wants to go to a restaurant for dinner with his friend near castello sforzesco in Milan. He is visually impaired and need to book a restaurant and inform him about the route map with voice to help him to reach the place. There are services available for completing these tasks such as *thefork*, a restaurant booking application and *AriadneGPS* that provides vocal information about the map for visually impaired people, but the user has to invoke these services separately which is time consuming and difficult for elders. Thus, there is a need to have a composite service that can provide his required information.

Use Case Scenario for Assistance: Mrs. Rossi wants to have a composite service which will give her assistance by setting an appointment to her doctor and will show the route map with the information to avoid the areas where air quality could worsen her symptoms as she has a problem of asthma. This can be possible by composing services like appointuit, Asthma health and Route Service like Moovit.

With the scenarios shown above, it is easy to understand that the different needs of elders can be fulfilled by composing available services or applications represented by APIs that have different functionalities. As these people know their own context and needs better than anybody else, and they often have real-time awareness of shifts in their respective domains, enabling them to tune applications to fit their requirements more closely is an obvious necessity. At this point, we realize the importance of using end-user development approach [32] to empower these end-users.

5.2.3 Technology Acceptance: Behaviour towards the Systems

Today's technologies should be accessible for all by anticipating the different needs of all sorts of people including elders, understanding their behaviors while they are consuming information, empathizing with them and whether it is convenient or frustrating for them along with their barriers. While the ageing process is different for everyone, all people go through some fundamental changes which affect the behavior towards the things or resources they are using. This is also true for learning and accepting new technologies that facilitate them an easier life. To

[1]https://www.w3.org/WAI/older-users/

assess the acceptance of a system or application Technology Acceptance Model (TAM) for elders have been used in several literature [22, 27, 112]. According to [13], to make the systems more acceptable we need to consider different ageing phenomena like less vision and hearing, limited motor control, preference of devices, experience and relationship with technology and cognition that is particularly relevant to design for the elderly: memory, attention and decision-making.

The constructs addressed in these models facilitate the assessment of the acceptance of a new system or technology by considering human behaviors. The considered constructs include social influence, perceived value, ease of use and safety. Another two important features of elders are *self-motivation* that implies the desire to complete a task and *learning capabilities* that implies the ability to learn about new technologies or systems. With having enough self motivation to complete a specific task and positive feelings towards using the system they intended to use it in real life according to their learning capability, ease of adoption and usage environment. All of these constructs are influenced by age, gender and experience. After analyzing the assessment procedure of this models, we realize that following aspects should be considered during the assessment of any system:

- Usability and the use of the system
- intention to repeated use
- positive usage experience
- willingness to recommend to others.

5.2.4 Perception of the Use of Internet, Web and Web Applications by Elders

The scenario described by research studies [51, 114] show that elder people, aged 58 to 77, are nowadays much more digitally aware than they used to be ten years ago. These studies also show that elder users are concerned about their independence and autonomy. Moreover, they need to know the function of the tool beforehand and to have continued support and confidence. Older people's adoption of ICT needs to be treated as more than merely a question of usability. Attitudes, experience of use, and perceived benefits are also key aspects that must be taken into account.

There is an increasing trend in using personal devices such as computers, mobile phones, tablets among elders. Paper [13] claims that that more people are now beginning to use the tablets because of the bigger size screen. Moreover, more than one third of these elderly people started to use smart phones. In future this number of the users of smart technologies is expected to grow since the producers of mobile technologies began to adjust them to elderly people's needs. Since the rapid growth of elder people, there is constant effort to prolong and maintain the active age of these elderly people, who want to lead active, fulfilling and quality life in terms of inclusion, socialization and independence. This can be achieved not only by a continuous support from their family members, but also with the help of modern information and communication technologies (ICT), which can assist elderly people in this process. The technological devices, which are primarily aimed at the elderly people, are also known under the common name gerontology because they try to meet the needs of ageing society [102]. In fact, present seniors are more and more involved in their use and these devices are becoming part and parcel of their everyday life [146]. Paper [47] claims that people aged 55 to 74 years are adapting to use of the Internet, mobile telephones, tablets, and gaming technologies nowadays. There are active elders who are accessing internet to use services for different purposes such as information search, communication with family and friends, use of social networks, or conducting online payments, online learning etc. Nevertheless, the exploitation of ICT by the majority of older people still seems to be too basic and we emphasis on the importance of exploiting their prior knowledge and capacities to be more innovative to support their behavior, performance and the situation they are in. Paper [15, 35, 64] summarize the main services the elderly people usually demand with respect to smart technologies:

- Health Monitoring needs: Due to the limitation of aged care resources, remote care services have become increasingly popular. Senior citizens often require special health vigilance, including monitoring of their physical and emotional conditions and the fulfillment of some medication agenda. Thus, it is necessary to adopt remote monitoring/vigilance of the elderly persons' health state, in order to improve the quality of the offered services.

- Personal Information needs: Most senior citizens spend most of their time at home with an occasionally visit for a short period. It is necessarily to provide advice / help located at their homes and at the places they visit, in some form of personal information maintenance, which includes medication and food constraints, personal interests and preferences,

- Social needs: It is very common that a senior citizen's relatives are engaged in jobs outside their homes and their friends are unable to visit on a daily basis. However, in general, relatives and friends want to offer help or keep in touch with each other. Thus, video conferencing and remote monitoring facilities would be useful.

- Leisure and Sales needs: Personal leisure (entertainment) is important for senior citizens in what constitutes their free time occupation. Obtaining books, hiring videos, ordering meals and groceries, playing games etc, allows the users to stay healthy mentally. ICT technologies will enrich the users' experience even if they are unable to go out regularly. Safety and privacy needs: This is considered as the most critical aspect for senior citizens.The user's activity must be monitored by using presence sensors, image sensors, etc, and be analysed with consideration to different scenarios.

Researches such as [35] also point out the need for lack of integrated application that can provide several features in an application that allows the users to track all his / her health readings. For example, the user is often required to use one application to record the heart-beat rate but open another application for recording the blood pressure. Thus, we emphasis on the need of having a system that can facilitate to create a composite application according to the end-users' specific needs.

5.2.5 Searching for Information on the Web: Role of Elders

Since the invention of Internet by Professor Tim Berners-Lee, a world of information is at our fingertips. We can explore related information about any topic. For example, watching an old film and want to find out if the actor appearing in it is still alive? Or perhaps watching a wildlife programme and want to learn more about the animals. In the past that would have meant consulting an encyclopedia or visiting a library. Now all we have to do is to go online and search for the information. Search processes in the World Wide Web play an increasing role in our everyday life. Users benefit from enormous quantities of available information and products on the Web, but the heterogeneous character of this mass of information and products is combined with a risk of retrieving suboptimal information: when users with incorrect prior knowledge search for information, it is likely that they may only follow those navigation paths that confirm their incorrect prior knowledge for a specific topic. on the one hand, the huge quantity of heterogeneous information on the Web makes it difficult for users to select adequate navigation paths for finding the best information. On the other hand, new Web technologies provide an opportunity to take advantage of the quantity of information and of the contributions of a myriad of Web users, and may support individual navigation and trigger learning processes. Previous studies have shown that searching for information on the web is more complex when the organization of the content displayed on the interface does not match the mental representations of end-users [18, 93] and when an individual lacks domain knowledge [50]. Concerning the last point, the existing literature, mainly studied involving young people, reported inconsistent results [16, 130]. Therefore, there is a need to examine the role of elder people in information searching and exploit these information in their daily activities. Ironically, using technology is also a potential barrier for older adults, because controls are typically difficult to see, operate, and remember. Moreover, memory capabilities decline with age, including the ability to recall recent actions. This deficit hinders older adults from completing tasks when interrupted or distracted. Paper [25] shows that world Wide Web is a major information source for elders but locating relevant information provided by the search engines is a major problem for them. Although poor metamemory appears to be a characteristic of older persons, experiments shows that there is no impact of this ability on information search activities performed by old end-users. Moreover,it shows that the Web experience can influence seniors' perceptions that means seniors with high Web experience became more critical,less confident and less enthusiastic than seniors with low Web experience. Paper [17] shows that an ergonomic version of a website is more acceptable to both the younger and older end-users considering searching of required information, cognitive load and usability satisfaction. Considering the above described literature, we aim to provide a guideline to design our proposed tool that follows the web accessibility guidelines to be accepted by all groups of end-users.

5.3 Requirements for Interacting with the Systems

As physical, sensory, and cognitive abilities decline with age the system should facilitate a front-end with the intuitive and user friendly interfaces and an easy interaction pattern with the system. Moreover, accessibility is the major concern while we are focusing on the systems for elders. The system should provide a possibility to adapt the user interfaces to the specific needs of elderly people, that implies the need to follow the inclusive design rules [33] at the front-end. Moreover, ease-of-use, simplicity and comfortable handling are required to satisfy the elder users. Also data security and privacy are essential for the elders to have trust in the web-based services, that can be supported by user authentication and control the access to personal data by implementing them in the back-end. As for decision making the elders are relying on trusted people or expert's opinion, connecting with social services to communicate with them needs to be considered.

5.3.1 Application Designs for Elders

Regarding designing good software applications for specific groups, designers should consider suitability of the software characteristics for general users and extend it to cope with limitation of specific users. Elder users are more interested on their interactions with the applications without knowing the back-end processes. In order to solve the problems of accessibility for elderly users who might have cognitive, physical or other limitations, interchangeable or adaptive interfaces are required. Such developments may also serve to provide younger disabled people with similar benefits, as well as for the wider population in general, in the context of the design for all concept. Moreover, to make new technologies more attractive HCI challenges include to provide privacy, allow informed choice and reduce complexity rather than increase the isolation currently felt by many older people [8]. Form the literature it is found that, the aging population presents a fundamental challenge to HCI in the need for socially dependable systems [25]. Socially dependable systems take account of social context, the need for sociability and are accessible to all who need them. There are considerable social and economic reasons why interface designers should rise to the challenge of designing interfaces which are usable by older people.

Table 5.1: Summary of usability recommendations from literature

Reference	**Title**	**Recommendations**
[53]	Creating senior-friendly web sites	Design, Layout, Content, Multimedia
[2]	Interface design guidelines for users of all ages	Layout and style, Colour, Text, General usability and testing, Accessibility and Disability, User customization.
[52]	Making your Web site senior friendly - A checklist	Designing readable text for older adults, Presenting information to older adults, Incorporating other media, Increasing the ease of navigation, Final check (usability testing).
[91]	Web usability for senior citizens	Presenting information and text, Presenting navigational elements and links, Search forms and results, Forms and data entry, Web address and homepage.
[19]	Designing web sites for older adults: Expert review of usability for older adults at 50 web sites	Interaction design, Information architecture, Visual design, and Information design.
[67]	Research-Derived Web Design Guidelines for Older People	Target (link) design, Graphics, Navigation, Browser window features, Content layout design, Links, User cognitive design, Use of colour and background, Text design, Search engine, User feedback and support.
[29]	Usability for older Web user	Avoid technical terms if possible, Identify links in a consistent and obvious way, The attention grabbing features on a page should be links, Visited links should change colour, HTML content where possible , Clear and concise content, ”Make the writing bigger” link and use of high contrast, Explicit instruction by using the imperative forms of verbs (e.g. ’Go to more details about XXX’)
[101]	Is a big button interface enough for elderly users?	Reduction of complexity, Clear structure of tasks , Consistency of information, Rapid and distinct Feedback, User support, Interface optimization, Customizable.
[149]	Successful and available: interface design exemplars for older users	Keep output messages as short as possible, Reduce choice wherever possible, Use mnemonic letters to indicate key press menu selections, Insert confirmatory statements wherever possible.
[1]	Improving WCAG for elderly web accessibility	Perceivable information and user interface, Operable user interface and navigation, Understandable information and user interface, Robust content and reliable interpretation.

We listed some important facts that should be considered during the design process of the application [36, 101, 143]:

- the design process must balance technical functionality and visual elements (e.g., mental model) to create a system that is not only operational but also usable and adaptable to changing user needs.
- user interface design should respond to different user needs, experience and capabilities.
- user interface design should match with user experience and expectations.
- designers should be careful about physical and mental limitations of software application users.
- software applications should provide user guidance and recover abilities from errors.
- a good interface should be consistent.

Many times, end-users judge usefulness of a system by only its interface. Thus, a poor interface can lead to catastrophic errors by user mistake. A lot of research has been carried out to understand elders' web accessibility needs that results in several HCI guidelines. These guidelines illustrate the recommendations to follow during the design of the interface of websites, web applications or services. We analysed different guidelines considering elder users, [1, 2, 19, 29, 52, 53, 67, 91, 101, 149], that have been presented in Table 5.1

These guidelines are provided to maximize the usability experience of the elders considering their age related declines and mental models. Although these guidelines are helpful for web designers and developers, there is a lack of awareness and repetitiveness of the guidelines [124]. Besides these guidelines there are suggestions regarding aspects that affects the elder web users presented in several studies.

When asked about suggestions from the users for making the Internet easier for seniors to use, the suggestions included [62]:

- Simpler pages; fewer buttons.
- Clearer Back/Forward option, including having the Back button in a new window closing it and returning the user to the window and page where it was linked from.
- Search to display result in associated grouping (like e-commerce sites).
- Fewer pop-ups.

Paper [48] also stresses the importance of simplifying the interface for older users new to computers and the Web and the results presented in [5] show that new elderly Web users benefit from simplified interfaces, especially while learning. Moreover, paper [38] suggested simplified interfaces as a solution to some forms of cognitive impairment by reducing on-screen complexity.

Paper [113] considered two aspects of form design and their impacts on elderly users (65-74 years):

- distinguishing between optional and compulsory fields, and
- the usability of check boxes, radio-buttons and list-boxes.

In a trial with the conventional asterisk and a form that clearly separated required fields from optional fields, all the participants expressed strong preferences for the "divided online form". Paper [25] found that locating relevant information among the search results from search engines was a major problem for elderly users. Paper [87] conducted a Web navigation study of both older and younger adults relatively new to the Web, found that the older users used the site map more than younger users and that the older users preferred to start from the home page when looking for information.

After analyzing all these studies we consider the following features as a guideline for our Web API composition environment:

- Privacy
- Set of controls to support elders (visual magnifier, sound control etc.)
- Clear structure of tasks
  - Page function unity (one page-one task)
  - Place confirmations every possible time
  - Distinct feedback from each actions
- Reduction of complexity
  - Avoid use of complex interactions
- Operable user interface and navigation
  - Clear navigation and location (search option, site map, descriptions of titles of pages/ results of search, indication of current location)
  - Clear purpose of a link
  - Sufficient time
- Understandable Metaphors and Icons
- Perceivable information and user interface (text size, color contrast, text style etc.)
- Understandable information and user interface (consistent navigation, instructions and input assistance, understandable language, error prevention and recovery for form)

5.4 Summary

In this chapter, we describe different requirements of a specific group of end-users, that is elder users who have certain potentials to act as end-user developers. We emphasis on their requirements for accessing the Web and Web applications followed by several existing studies, so that we can leverage this knowledge in designing an appropriate tool for our proposed composition platform. We analyse different interface design guidelines and make a list of usability recommendations that are appropriate for our proposed solution. In the next chapter, we will present the technical solutions that will support the back-end processes of Web API Composition. We will present our prototype implementation in chapter 8, following the recommendation outlined in this chapter.

Chapter 6

Interoperability through RESTful Interactions

As addressed in the problem definition in chapter 3, we realize that the presence of heterogeneous communication protocols and interfaces prevents from giving self-managing capabilities to service systems. To resolve this issue, in this chapter we describe our approach [81] to create a shared platform that fosters the definition of services that can be easily integrated and controlled. To accomplish this we exploit the widely adopted HTTP protocol. Such services are provided with RESTful interface and interaction style to gather data and control behavior of sensors to support the development of sensor services integrated with other services in pervasive systems.

6.1 Basic Concepts and Principles

Over the last decade, the Web has grown from a large-scale hypermedia application for publishing and discovering documents (i.e., Web pages) into a programmable medium for sharing data and accessing remote software components delivered as a service. The need of global availability and sharing of huge amount of information through various kinds of heterogeneous devices and services has changed the reference scenario for the development of Web scale applications. The consequent growing complexity and increasing request of adaptive services has made manual management impractical. A possible answer toward self-management is to make services smarter by achieving awareness of the target things' or the applications' physical environment or situations to respond proactively and intelligently. To tackle the problem a first effort should be the definition of a common protocol to foster automated interoperability. As a matter of facts, the HTTP protocol has been used as a universal mean for tunneling messages in business-to-business scenarios, but quite often to support additional protocols, such as WSDL/SOAP for Web Services or OGC/SOS/SPS for sensors, that are usually not interoperable. In this chapter, we investigate the correct and complete use of the HTTP protocol to publish, manage, and operate services on the Web by fully exploiting the REST (REpresentational State Transfer) principles [30]. Today, more and more services published on the Web are claiming to be designed using REST, but actually missing some of the features, like the hypermedia control, which is crucial to automate self-management operations.

6.2 Illustrative Example: RESTful Interactions between Heterogeneous Web Services

Nowadays, the number and variety of available services on the Web enable for the definition of sophisticated composed services, which can take great advantage from pervasive computing and sensor Web to give users seamless assistance in daily activities. In pervasive computing environments, different devices often use different technologies, which make the communication complex and sometimes even troublesome. Sensor Web has a central role in addressing the sources of information that need to be dynamically integrated into systems. The Sensor Web Enablement architecture of the Open Geospatial Consortium (OGC) [10] has been developed to enable flexible integration of sensors into any type of software system using standards: the Observations and Measurements and Sensor Model Language specifications define the exchanged sensor data; the Sensor Observation Service (SOS) gives pull-based access to observation data [12]; the Sensor Planning Service (SPS) tasks sensors [122]. We want to point out, through the discussion of a use case, the problems that arise from the diversity of the involved services, and show how the communication burden can be transferred from the user to self-managing pervasive systems. Then, we investigate the advantages of the RESTful approach as a mean for providing a common interface for services communication and control.

6.2.1 Use Case Scenario

We consider two versions of a scenario in which a user, let's say John, wants to go to watch a movie looks for route directions to available parking spaces close to the cinema hall. In the first version, the user is asked to interact with different Web Services of different types, such as Web Form-based, SOAP, API and RESTful Web services. As we have included sensors in our scenario, different OGC compliant services such as SOS and SPS are invoked. In the second version, a self-managing composite service minimizes the user interaction by transferring the control from the user to the service. Self-management will be achieved by providing sensors and services with RESTful interfaces to make interaction and control similar to the one already in use on the Web and therefore already familiar to both users and developers.

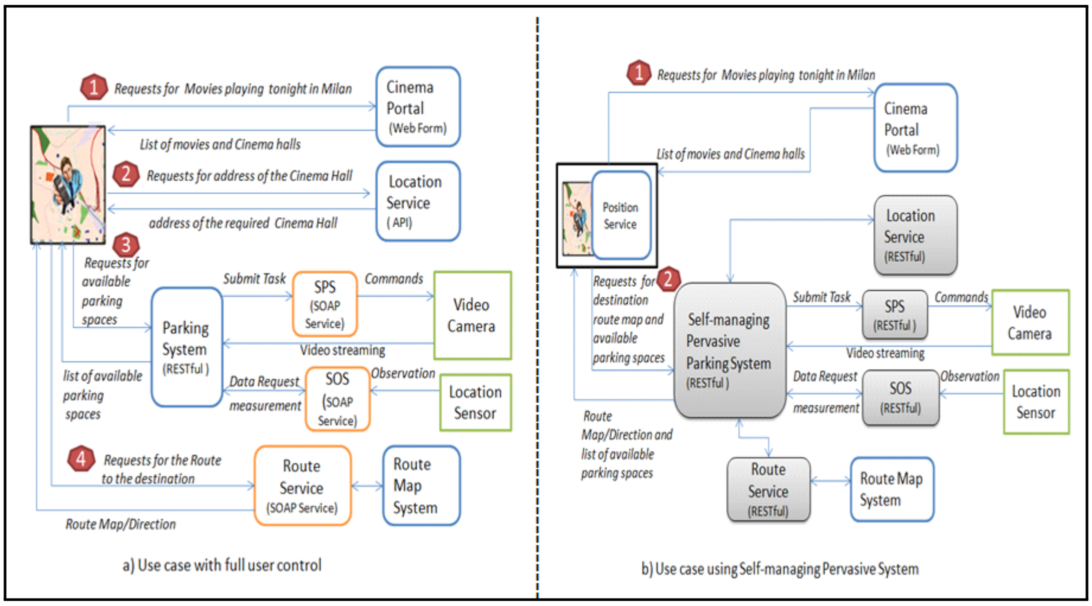

1 Requests for Movies playing tonight in Milan
Cinema Portal (Web Form)
List of movies and Cinema halls
2 Requests for address of the Cinema Hall
Location Service (API)
address of the required Cinema Hall
3 Requests for available parking spaces
Parking System (RESTful)
Submit Task
SPS (SOAP Service)
Commands
Video Camera
Video streaming
list of available parking spaces
Data Request
measurement
SOS (SOAP Service)
Observation
Location Sensor
4 Requests for the Route to the destination
Route Service (SOAP Service)
Route Map System
Route Map/Direction
a) Use case with full user control
Position Service
Location Service (RESTful)
Requests for destination route map and available parking spaces
Self-managing Pervasive Parking System (RESTful)
SPS (RESTful)
SOS (RESTful)
Route Map/Direction and list of available parking spaces
Route Service (RESTful)
b) Use case using Self-managing Pervasive System

Figure 6.1: Use case scenario

6.2.2 Use Case Scenario with User Control

As depicted in Fig. 6.1 (a), first John searches for movies by filling up the Web form of the Cinema Portal, and gets the list of movies meeting his search criteria, with the corresponding cinema hall name. Then, John chooses a cinema hall and invokes the Location Service API to get the address. Next, he invokes the Parking System, with RESTful interface, to get information about the available parking spaces close to the selected cinema hall. The Parking System interacts with the SOS and SPS services to get the observations from the video cameras and location sensors installed in the parking lots, and makes decisions about the available parking spaces. Finally, John chooses one parking lot from the list of the available ones, indicates his current position, and invokes the Route Service, through SOAP messages, to get directions on how to reach his destination.

6.2.3 Use Case Scenario Based on a Self-managing Approach

In this version (Fig. 6.1 (b)), the scenario has been changed to minimize the user control and add self-manageability to the system. The user invokes only two services: the Web Form-based Cinema Portal and the Self-managing Pervasive Parking System. Further, as shown in Fig. 6.1 (b), the user has the Position Service in his mobile, which is able to get automatically the position of the user and deliver it to other services. At first, John interacts with the Cinema Portal to get the list of movies of his interest, as before. Then, he selects a cinema hall and invokes the Parking System to get all the information needed to reach the destination. The Parking System interacts with different services: (i) invokes the Location Service to get the location of the selected cinema hall, (ii) invokes SOS and SPS to get observations about parking spaces, and decides which are the available parking spaces near to the cinema hall, (iii) gets the route direction for the nearest available parking spaces by interacting with the Position and the Route services, and (iv) delivers the cinema hall route direction to the user with maps.

6.3 Common Interface for Services Interactions: The RESTful Approach

In order to increase the system interoperability, we have designed the Self-managing Pervasive Parking System according to the Resource-Oriented Architecture that is devoted to manage distributed, heterogeneous resources in which client applications interact directly with the resources, by following the REST principles [30]:

1. Resources should be identified properly using URIs, so that each resource is uniquely addressable.

2. Uniform interfaces should be provided through the use of a standard application level protocol. In this way, the operations to be applied on resources are external and they have well known semantics [99].

3. Resources are manipulated through their representations, since clients and servers exchange self-descriptive messages with each another. A resource can have multiple representations that follow a standardized format or media type and can be negotiated with the Web server. Representations convey the state of the client's interaction within the application and contain hyperlinks that allow clients to discover other resources or change the state of the current resource.

4. Interactions are stateless since servers only record and manage the state of the resources they expose, i.e., client sessions are not maintained on the server. This increases the decoupling between client and server.

5. Hypermedia is the engine of application state, i.e., the application state is build following hyperlinks according to the navigation paradigm. Therefore, the application state is not known a priori, but it is built based on user navigation.

Further, we expose data produced by the services according to the Linked Data paradigm. Linked Data Design defines[1] the following rules for exposing structured data on the Web:

- use URIs to identify data as names,

[1] http://www.w3.org/DesignIssues/LinkedData.html

- use HTTP to look up those names,
- provide useful information about URIs using standards, and
- links to other URIs, so that they can discover more things.

Most of the technologies used in the first version of our use case (6.1 (a)) do not follow the REST principles. Consequently, they compromise the interoperability of services and do not facilitate computer-to-computer interactions in the context of self-managing pervasive systems. In particular, the Route, SOS and SPS services communicate through SOAP messages. Thus, the HTTP methods are used as transport protocol while the application protocol is domain specific and the operations invoked by the user lay on the message envelope. Such communication pattern tunnels all the requests to a single URI that identifies an endpoint. HTTP GET and POST are the most-in-use methods but their semantics are not maintained, i.e. GET is used to invoke operations on server side that modify resources state, and therefore, it is not possible to optimize the network traffic by using caching mechanisms. Different is the case of the Web form-based Cinema Portal. The user interacts with different URIs via an html form, using again GET or POST possibly with a different semantics: URIs encapsulate server-side information, like operation names and parameters, revealing implementation details to the user. Such an approach enforces the coupling between the client and the server: if the server implementation changes, then the old URIs become invalid (operation and/or parameter names may change). The Location service publishes an API but interactions with the service are not hypermedia-driven since resources representation does not contain hypermedia controls and the user advances to the next state using some out-of-band information . The pervasive computing system is expected to be seamlessly adapting, in a fully autonomic way, to different operational conditions to fulfill the user requirements. Autonomous actions need to be performed by enabled devices, sensors and/or services with different levels of capabilities.

The Pervasive Parking System needs to access and control services that use different technologies in the communication which make interactions troublesome. This problem can be addressed by employing a common architectural style for implementing the involved interfaces. We propose to adopt the REST approach because interoperability is fostered by the use of standard technologies, the stateless RESTful interactions support scalability, and hypermedia controls reduce coupling between components by driving clients' interaction. Moreover, REST principles provide the opportunity to reuse and generalize the component interfaces, reduce interaction latency, enforce security, and encapsulate legacy systems by using intermediary components.

6.4 Design of a RESTful Interface for OGC Services

In this section, we describe how to design RESTful services that are compliant with the OGC's SOS and SPS standards. We focus on activities that allow data sensor requestors to submit sensor tasks and retrieve the generated observations. These are the SPS operations GetCapabilities, DescribeTasking, DescribeResultAccess, Submit, Update, Cancel and GetStatus, and the SOS operations GetCapabilities, DescribeSensor and GetObservations. The proposed paradigm enables computer-tocomputer interactions since, in the context of sensor data it is a common need to have automated processes elaborating even raw data to proprietary formats.

The overall goal requires first to identify the structural elements of a RESTful interface because these will become the building blocks for RESTful services. The retrieval and managing of sensor data as resource representations requires the identification of (i) the resources of interest, and (ii) the permitted hypermedia-driven interactions of the consumer with the service offering the sensor data. We explore these two dimensions in terms of the URI scheme used to describe resources, the HTTP idioms (methods, headers, status codes) used to interact with the service, the domain application protocol, and the media type for hypermedia-driven interactions.

6.4.1 Sensor Data as Resources

In SOS and SPS services, resources result from the computations applied to data describing capabilities of services exposing sensor data, sensors as well as their tasks and their observations. The main data classes of objects retrieved are the following: SOS Capabilities provides metadata about a SOS service in terms of (i) parameters that may be used to filter the retrieval of observations, (ii) observation offerings used to further organize observations into sets, and (iii) set of sensors associated with the service; SPS Capabilities provides metadata about an SPS service in terms of sensors and phenomena that can be tasked; Sensor provides the highest level of detail of sensor metadata; Task provides information about the status of the tasking request and the task itself; Observations groups a set of observations retrieved using the same criteria; Observation associates a retrieved value with the sensor providing it, the offering it belongs to, the time period in which it was taken, etc. The above classes are not enough to describe all the data objects involved in the interactions with SOS and SPS services, and thus, we introduce: the FeatureOfInterest to describe the observed stations; the Observation Offering to organize observations; the ObservedProperty to describe the phenomenon, i.e. temperature, we want to observe; the Time to indicate the time period of observations; and the Procedure to describe the method used to observe a phenomenon.

6.4.1.1 URI Scheme

We assign URIs to the objects of the above classes using the conventions proposed in [107] and used in [58, 142]. The base URI for every class has the form: http://my.host/class. Therefore, the URI http://my.host/observations represents a collection of all the observations published by the service. Further segments following the base URI refer to additional criteria to be used for the retrieval and correspond to parameters to be applied to the GetObservations SOS operation. We have identified the following situations in the parameters:

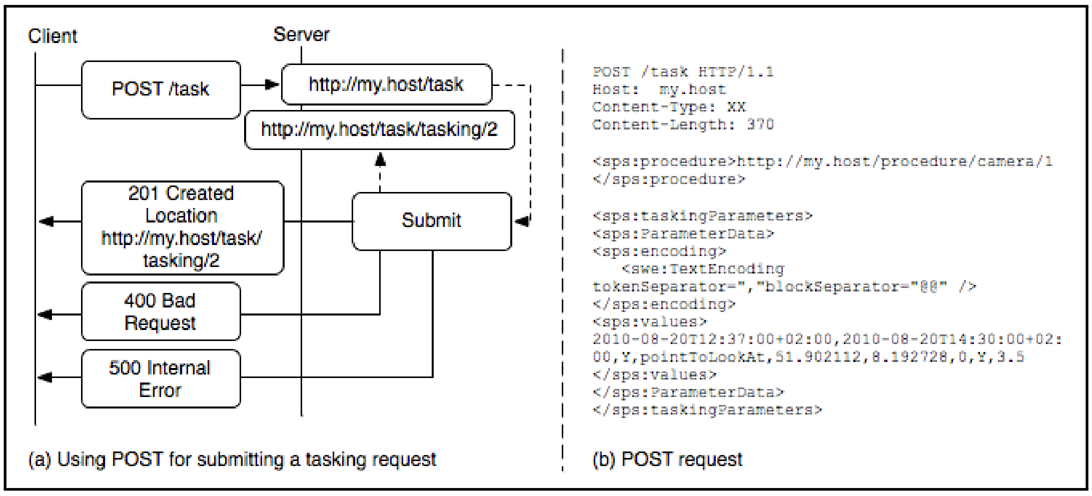
- If the operation call requires multiple parameters, then two segments are appended to the URI for each parameter: the first segment indicates the data class corresponding to the parameter name, while the second one indicates the parameter value. Although, this is a strict order for the segments of one parameter, the segments of distinct parameters can be appended to the URI in an arbitrary way. The retrieved observations are those that satisfy all the criteria indicated in the URI. The http://my.host/observations/sensors/urn:ogc:object :Sensor:MyOrg:13/observedProperty/urn:ogc:def:property: MyOrg:AggregateChemical Presence points to the collection of observations about urn:ogc:def:propertyMyOrg:Aggregate ChemicalPresence taken by the sensor urn:ogc:object:Sensor:MyOrg:13.

- If a parameter has multiple values, then, since order is not relevant, the semicolon is used to separate the values in the corresponding segment. The retrieved observations are those that satisfy the indicated criteria for at least one of the values. The http://my.host/observations/sensors/urn:ogc:object:Sensor: MyOrg:13;urn:ogc:object:Sensor: MyOrg:12/observedProperty/urn:ogc:def:property: MyOrg:AggregateChemicalPresence points to the collection of observations from the urn:ogc:object:Sensor:MyOrg:13 or urn:ogc:object: Sensor:Org:12 sensors.

- If a parameter has a range of values, then, since the order is relevant, the comma is used to separate the start value of the range from the end value in the corresponding segment. The http://my.host/observations/sensors/urn:ogc:object: Sensor:MyOrg:13/eventTime/2011-04-14T17,2012-04-14T17,tm:rel:between/ points to the collection of observations taken from April 14th 2011 at 5pm and April 14th 2012 at 5pm. The time strings are encoded according to ISO 8601.

- If a parameter has structured values, then, since the order indicates the meaning of the value and thus, it is relevant, the comma is used to separate the values in the corresponding segment.

The above rules define the meaning of the URI segments and they are applied to all classes for assembling the URIs of the data objects defined by the class. The first segment after the service host indicates the class whose objects are to be retrieved. The successive segments indicate filtering conditions to be applied to the underlying retrieval mechanism. Because of the different computations that may be applied to retrieve the same resource, we have different URIs representing the same resource. This facilitates the resource access, but it could also be troublesome, if clients are not able to distinguish whether different URIs refer to the same resource. We follow the convention stated above and for each resource we keep an URI that serves as a reference for all the other URIs. Then, we explore hypermedia mechanisms to insert this reference URI in the response every time the client requests other URIs.



```
POST /task HTTP/1.1
Host:  my.host
Content-Type: XX
Content-Length: 370

<sps:procedure>http://my.host/procedure/camera/1
</sps:procedure>

<sps:taskingParameters>
<sps:ParameterData>
<sps:encoding>
   <swe:TextEncoding
tokenSeparator=","blockSeparator="@@" />
</sps:encoding>
<sps:values>
2010-08-20T12:37:00+02:00,2010-08-20T14:30:00+02:
00,Y,pointToLookAt,51.902112,8.192728,0,Y,3.5
</sps:values>
</sps:ParameterData>
</sps:taskingParameters>
```

Figure 6.2: POSTing a tasking request

6.4.2 Hypermedia-Driven Interactions with Sensor Services

SPS and SOS services are complementary: the former schedules tasks for collecting sensor data, while the latter publishes the collected data. In particular:

- tasking requests are created when the user makes a submission;
- tasks are created by the service when a tasking request has been accepted;
- a user may retrieve tasks as well as tasking requests, to know their status;
- while a task is in execution, it can be updated and/or cancelled;
- during the execution of a task, observations are created and published;
- observations may be retrieved at any time after their publication.

6.4.2.1 HTTP Idioms

We distinguish the HTTP methods that may be used for the invocation of the above actions. Tasking requests are created with POST, retrieval of resource representations is achieved with GET, and tasks are updated with PATCH and cancelled with DELETE. We use POST instead of PUT for creating tasking requests because the service and not the user will create and associate an URI with the newly created resource. We use PATCH instead of PUT for updating tasks because the service allows incremental and not overall modifications.

In Fig. 6.2 we exemplify the HTTP request for submitting a tasking request. The request is composed of the POST method, the path that will serve the request at the server and the payload containing the tasking parameters. Upon reception of the request, the control is passed to the Submit operation, but the user remains unaware of this since the operation is inside the server

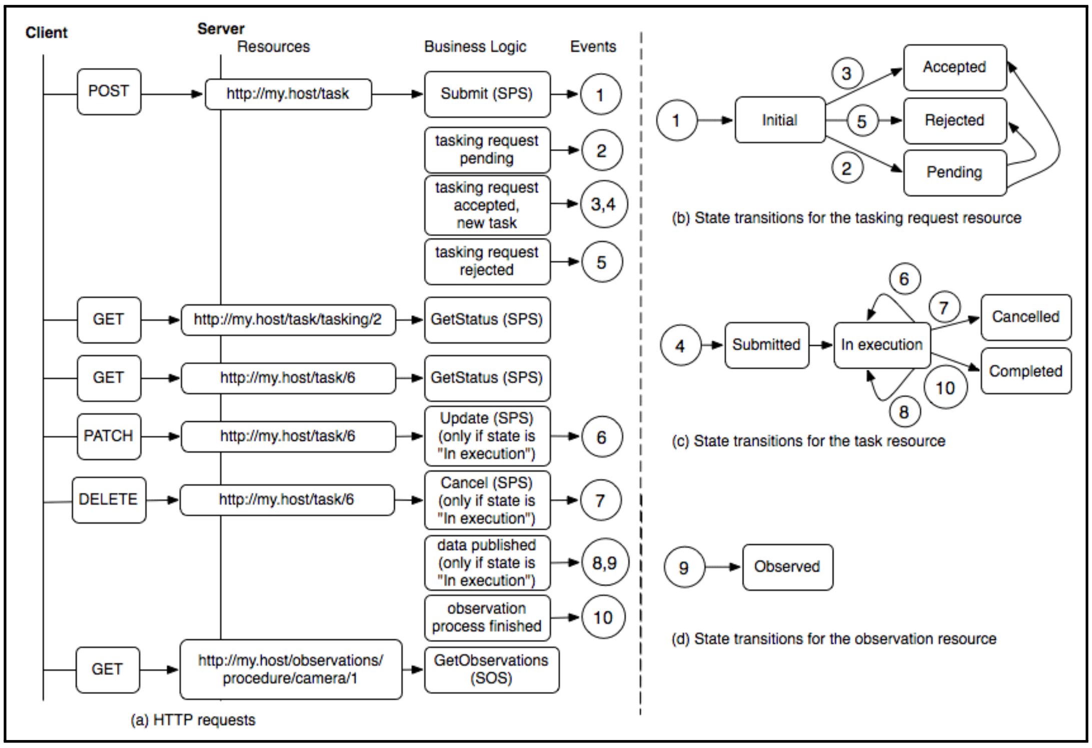

Client
Server
Resources
Business Logic
Events
POST
http://my.host/task
Submit (SPS)
1
tasking request pending
2
tasking request accepted, new task
3,4
tasking request rejected
5
GET
http://my.host/task/tasking/2
GetStatus (SPS)
GET
http://my.host/task/6
GetStatus (SPS)
PATCH
http://my.host/task/6
Update (SPS) (only if state is "In execution")
6
DELETE
http://my.host/task/6
Cancel (SPS) (only if state is "In execution")
7
data published (only if state is "In execution")
8,9
observation process finished
10
GET
http://my.host/observations/procedure/camera/1
GetObservations (SOS)
(a) HTTP requests
Initial
Accepted
Rejected
Pending
(b) State transitions for the tasking request resource
Submitted
In execution
Cancelled
Completed
(c) State transitions for the task resource
Observed
(d) State transitions for the observation resource

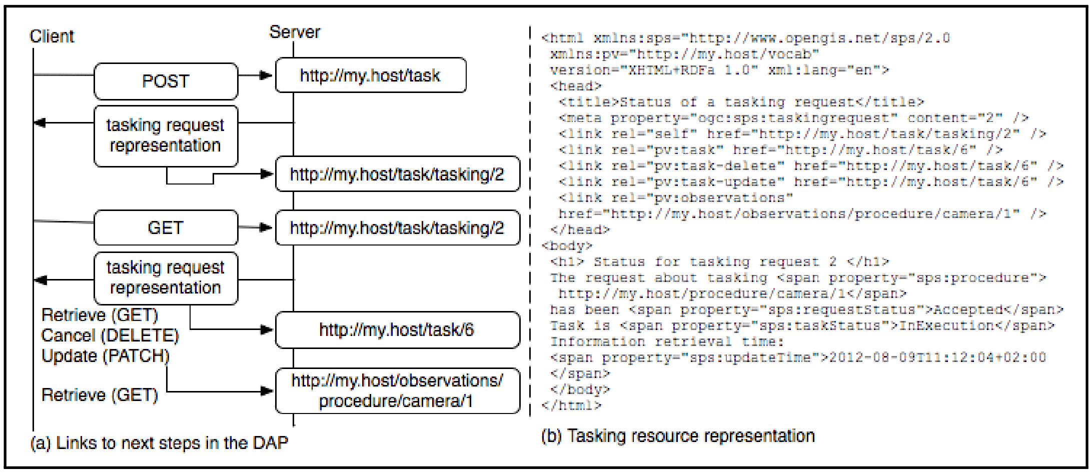
Figure 6.3: (a) DAP and ((b)(c)(d)) resources state transitions

boundaries. The operation creates a tasking request resource and sends the HTTP response to the user. Upon successful creation of the resource, the response status code is 201 Created, and the returned URI identifies its location. In the meanwhile, the system decides whether the requested task will be accepted, rejected or pending. The URI of the tasking request resource identifies also the task to be created and it will be used to GET, PATCH and DELETE the task on the server. Other status codes indicate: (i) a malformed request, and (ii) server failures that prevent the server from fulfilling the request. Similar are the HTTP requests for the remaining actions.

6.4.2.2 The Domain Application Protocol

Possible interactions a single user may have with both SOS and SPS services constitute an overall protocol for tasking sensors and getting their observations. The partial or complete execution of this protocol changes the state of the resources involved in the communication like tasking request, task and observations. Our method follows the methodology in [142]: resources are the central aspect in the service implementation and the user interactions drive their life cycle.

In Fig. 6.3 (a), we depict the HTTP requests of possible interactions and the events that will fire the state transition for resources. The user POSTs a tasking request, and the SPS Submit operation creates the tasking request resource that enters the Initial state (as shown in Fig. 6.3 (b)). Internal business logic decides whether the tasking resource's state will transition to


Client
Server
POST
http://my.host/task
tasking request representation
http://my.host/task/tasking/2
GET
http://my.host/task/tasking/2
tasking request representation
Retrieve (GET)
Cancel (DELETE)
Update (PATCH)
http://my.host/task/6
Retrieve (GET)
http://my.host/observations/procedure/camera/1

(a) Links to next steps in the DAP

```
<html xmlns:sps="http://www.opengis.net/sps/2.0
 xmlns:pv="http://my.host/vocab"
 version="XHTML+RDFa 1.0" xml:lang="en">
 <head>
  <title>Status of a tasking request</title>
  <meta property="ogc:sps:taskingrequest" content="2" />
  <link rel="self" href="http://my.host/task/tasking/2" />
  <link rel="pv:task" href="http://my.host/task/6" />
  <link rel="pv:task-delete" href="http://my.host/task/6" />
  <link rel="pv:task-update" href="http://my.host/task/6" />
  <link rel="pv:observations"
  href="http://my.host/observations/procedure/camera/1" />
 </head>
<body>
 <h1> Status for tasking request 2 </h1>
 The request about tasking <span property="sps:procedure">
  http://my.host/procedure/camera/1</span>
 has been <span property="sps:requestStatus">Accepted</span>
 Task is <span property="sps:taskStatus">InExecution</span>
 Information retrieval time:
 <span property="sps:updateTime">2012-08-09T11:12:04+02:00
 </span>
 </body>
</html>
```

(b) Tasking resource representation

Figure 6.4: Hypermedia-aware resource representation

Pending, Accepted or Rejected state fired by the corresponding events 2, 3 and 5. In case the tasking request is accepted, a new task resource is created, and the event 4 fires the entering of the task resource to the Submitted state and then automatically, to In execution state (Fig. 6.3 (c)). After the submission, the user may GET the status of the tasking request and/or task resource. These interactions do not have any server-side effect. As long as the task resource is In execution state, the user may PATCH modifications or DELETE the resource firing the events 6 and 7, and the publication of the collected sensor data is enabled. The latter activity fires the newly created observation resource to the Observed state (Fig. 6.3 (d)) and the user may GET its representation. Once the collection process is finished, the task resource transitions to the Completed state.

6.4.2.3 Hypermedia-aware Media Type

In order to enable the user to create tasking requests and get observations without the need to explore any out-of-band information, we need to provide him with the entry point http://my.host/task where he may POST a tasking request, and then insert hypermedia controls in the HTTP responses so that link relations drive the user in the protocol described above.

In Fig. 6.4 (a), we exemplify this mechanism with the HTTP response to the POST tasking request. The response contains both the tasking request resource representation, and links to further steps in the protocol. If the tasking request is pending, then the response provides a link for inquiring again the current tasking request and the task to be generated (it has the same URI as the tasking request). A subsequent user GET request for the tasking request representation, results to a different response since, in the meanwhile, the tasking request has been accepted and the task is in execution.

The representation contains further links for retrieving the task resource, requesting modifications and canceling it, as well as links for getting the observations produced so far. The media type for resource representations must understand the semantics of these links in order to enable their automatic interpretation. We have used XHTML+RDFa in order to enable both human-to-computer and computer-to-computer interactions. RDFa extends XHTML with metadata that have the form of triples: subject-predicate-object. Link elements in Fig. 6.4 (b), convey the location where the task submitted by the current request maybe (i) retrieved, (ii) updated, and (iii) cancelled, and the location where observations may be retrieved. The meta element informs that the current context represents an SPS tasking request with id 2. The XML attribute property annotates semantically html elements so that a software agent understanding the SPS definitions can extract the enclosed information.

6.5 Transformation of Legacy Services to RESTful Services

Considering the scenario in Fig. 6.1 we developed a Web service using JAVA Servlet, HTML and MySQL that represents the Cinema portal. Using this service the user can search for movies in a specific place and can check the available seats in a cinema hall with other related information. He can book the ticket and also can make the payment.

To do in-lab experiment on our proposed approach, we employ a RESTful interface that we termed as "REST wrapper" to make this cinema Web service interoperable with other RESTful Web Services in our scenario. We used Jersy [2](a JAX-RS[3] implementation) to build the REST wrapper by following [142]. Then we bind together this legacy cinema Web service and the REST Wrapper. By employing this RESTful interface the service is able to provide hypermedia driven interactions. Using JAX-RS, we declare a set of methods, to which the framework routes HTTP interactions.

In our example, to buy a a ticket the user has to follow the following actions:

- Invokes the Cinema service for retrieving information about free seats using GET;
- Requests for a booking by POSTing a request;
- Updates or cancels the booking by using PATCH/DELETE;
- Retrieves booking information, to know the status by using GET;
- Requests for a payment by POSTing a payment request;
- Retrieves ticket ordering information, to know the status or get the ticket by using GET;

We implemented Resource classes BookingResource, PaymentResource and TicketReceiptResource. Inside the service, resources act as controllers for workflow activities, passing through business information extracted from the representations and marshalling results and exceptions

[2]https://jersy.dev.java.net
[3]https://jax-rs-spec.java.net/

into HTTP responses. Workflow activities implement the individual stages of the cinema services's workflow in terms of resource life cycles: creating booking, updating booking, canceling bookings, creating payments, and delivering completed ticket orders to customers. They are responsible for changing the state of the underlying domain objects, which in turn are persisted in repositories. Each activity knows which activities are valid. for example, if payment succeeds, the valid next step is to ask for a ticket or to check ticket status.

Use of these HTTP methods and the representation (application/vnd.restcinema+xml) provides a way to drive users from one state to the next state without knowing the system a priori. For example, while the request from the client side is made for the information about free seats using GET, the service provides the available information and also the links for booking in the representation. Then the client can select the booking option to reserve a seat in a specific date and time. This request is made by using POST method and upon reception of the request, the service creates a booking request resource, decides whether the booking will be accepted, rejected or pending, and sends the HTTP response to the client. Upon successful creation of the resource, the response status code is 201 Created, and the Location header identifies its URI. The representation provides the links for updating, canceling or getting the information about booking by using PATCH, DELETE and GET respectively and also provides a link for making the payment. This request is done by using POST that creates a resource named as payment resource. After successfully creating this resource the representation is provided with a link for getting the ticket using GET. Similar HTTP requests are used for actions to fulfill the user needs in the invocation of other services.

During this implementation, we realized that binding the existing Web services into RESTful interface is very complex and requires a lot of manual works. Moreover, we identified following issues:

- server client are tightly coupled.
- Existing services cannot be easily updated to support HATEOAS.
- Inappropriate for Clients when a service is changed.
- Inaccessible source code.
- JAX-RS does not support PATCH, so has to implement it.
- The Legacy Cinema Service has some internal Id in the database which can not be used in the URL of The REST Cinema Service.
- HTML5 form element supports only GET and POST,
- for other HTTP requests have to create browser's object (XMLHttpRequest) for sending HTTP requests instead of HTML forms.

6.6 Summary

In this chapter, we have made a step towards the development of Self-managing Pervasive Systems by discussing the definition of RESTful interfaces to existing services, and, in particular to sensor services. We exploited the widely adopted HTTP protocol to create a shared platform and show how the services can be easily integrated and controlled by using our approach. The adoption of such a widely used communication standard enables for interoperability, which is the basic requirement to give services the capabilities to adapt their behavior. We have shown how Web services can become smarter and deliver more complex functionality by gathering information from sensors and traditional services with minimal human intervention, using inter-linkage of sensor data with hypermedia controls. We have done an in-lab experiments by implementing a REST wrapper and bind it with a legacy Web Service. Through this experiment we have learned that, it is possible to provide RESTful interactions among heterogeneous Web services by providing RESTful interfaces where the users are guided through the hypermedia controls. However, there are some implementation issues, we believe these can be resolved by using new features given by new frameworks. For example, when we use Swagger to document a REST API, we can use existing @PATCH annotation defined in the io.swagger.jaxrs package. In this way, the complexities become reduced and this approach becomes more adaptable.

CHAPTER 7

ENRICHING API DESCRIPTIONS BY ADDING API PROFILES THROUGH SEMANTIC ANNOTATION: AN APPROACH FOR FACILITATING WEB API COMPOSITION

In recent years several description tools and formats have been introduced for describing REST Web APIs both in human and machine readable formats. Although these descriptions provide functional information about the APIs (e.g. HTTP methods, URIs, model schema, etc.), the information that qualifies the properties of APIs (e.g. classification of input arguments and response data) is missing. We envisage that providing a complete set of information to the users will facilitate the composition of APIs to fulfil users' specific needs.

By analyzing the current state of the art in Web API Descriptions and Semantic Annotations presented in chapter 4, we realize that, although there are solutions with semantic capabilities, most of them fails to add semantic annotations automatically or semi-automatically. Moreover, advanced technical skills are needed to manage semantics and compose different Web APIs, which reduce the number of potential users of such solutions. The goal is to enhance actual API descriptions by creating a simple description format to annotate properties at semantic level to support semi-automatic composition. To achieve this goal, we propose an extension of the Open API Initiative (OAI) specification to create comprehensive descriptions. The approach focuses on the emerging concept of API Profiling to add descriptive information of data semantics by addressing Dublin Core Application Profile (DCAP) guidelines.

In this approach, the professional developers are acting as an intermediary between the end-users and the actual service providers. Their tasks include the enrichment of the API descriptions and creation of composite Web APIs by defining the composition patterns, to facilitate a user-driven composition to be dynamically preformed by the end-user developers. The following sections describe this approach in details as presented in our paper [79].

7.1 Basic Concepts and Principles

As web-enabled software becomes the standard for business processes, the ways organisations, partners and customers interface with it have become a critical differentiator on the market. Therefore, the ability to provide appropriate and complete Web API descriptions to let users discover applications that satisfy a set of requirements and compose applications to fulfil more complex users' needs is critical for the success of any organisation. Although the process of implementing APIs has become common practice, meta-level API definition and implementation have yet to be settled to set widely-accepted standards. Today, description formats, such as Open API Initiative (OAI) specification[1], also known as Swagger[2], RAML, API Blueprint[3], are available to describe implementation details including resources, access points, status codes and input arguments [88]. These description formats are created by following the API-first approach[4] and using a meta-language based on XML, JSON or YAML. Moreover, a set of tools have been developed to create API descriptions interactively: such tools can auto-generate server-side code, testing options for different HTTP methods, or even fully functional API Clients (e.g. Swagger Codegen[5]). These formats and tools are mostly human-driven and lack supports for detailed information that qualifies the properties of an API (e.g. classification of input arguments and response data). Moreover, these formats may meet the requirements of developers to complete simple tasks, but they are inefficient in advanced API discovery or API composition due to the lack of machine processable semantics [136]. Moreover, such formats should be made easy to understand when the target users include high-level business experts or specific groups of people (e.g. the elderly, people with disabilities, etc.) who do not have specific programming expertise. We name these users as "*end-user developers*". Several studies [23, 89] show the need of interactive documentation that provides flexible navigation alternatives with a comprehensive set of information to support a wide range of users. As users' background influences how they navigate the documentation, there are barriers for *end-user developers*, due to inconsistent and very technical terminology use. The final goal of our work is to develop descriptions that can be (semi) automatically composed by developers to support end-user composition of APIs, therefore we address the following questions:

- Are there widely adopted approaches, tools or standards for creating machine processable API descriptions with semantics?
- What are the missing features in current API Description formats to aid composition?
- How can existing approaches be improved by adding semantics to API Descriptions to facilitate (semi) automatic user-driven composition?

[1]http://openapis.org/specification
[2]https://www.swagger.io
[3]http://raml.org/, https://apiblueprint.org/
[4]http://www.api-first.com/
[5]http://swagger.io/swagger-codegen/

In this work, we consider the composition of REST Web APIs by adding machine readable semantic descriptions. The approach is to describe properties with semantic meaning by linking to concepts in shared vocabularies.

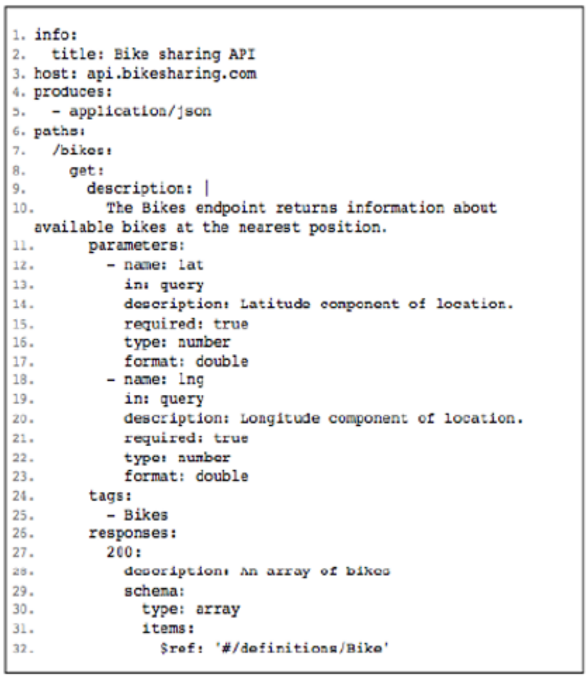
In the real world, if developers want to compose APIs, they may search directories such as Programmable Web[6], and understand the meaning of involved data, e.g. that *address* means *city* and *street*, or *latitude* and *longitude*, but a machine agent is unable to understand the meaning without a shared representation of property semantics. The use of links to concepts in shared vocabularies that allow a machine agent to compare and compose the actual data can address this issue. We propose to exploit API profiles to provide descriptive information about the contents of the response according to the Application Profile[7] approach described as a set of metadata elements, policies, and guidelines defined for a particular application.

In this work, we evaluate the current approaches to create API descriptions and make a proposal to include additional qualifying information. Our goal is to enhance interoperability and composition by creating a standard description format that correlate properties at semantic level. To achieve this goal, we propose to include API profiles with the API descriptions created by following the OAI specification.

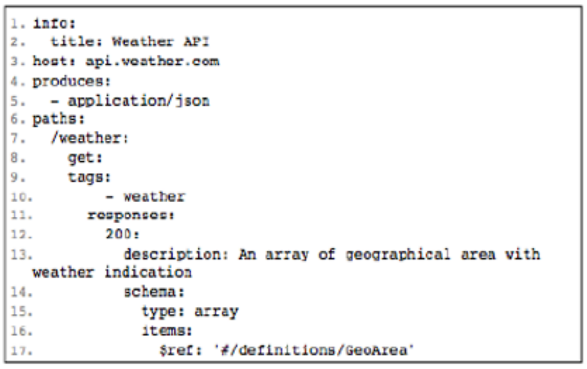
We adopt the Dublin Core Application Profile (DCAP) guidelines[8], to share data semantics in a specific representation format. We propose the use of a (semi) automatic method for adding annotation, TableMiner [150], which is a semantic table interpretation method to extract the most appropriate concepts from shared vocabularies in a (semi) automatic way by using context information. This approach classify table columns and disambiguate cell contents following different algorithms. This approach considers two types of contexts, one is defined as “in-table context”, including column header, column content and row content, and another one is “out-table context”, which could include semantic mark-up already inserted in a web page, the web-page title, paragraphs and table captions through following steps:

- first, it adopts a bootstrapping, incremental approach to interpret columns with at least 51% of non-empty rows and with mostly named entities;
- then, a forward-learning process uses an incremental inference with a stopping algorithm that makes a first semantic association with the contents of columns, followed by a process of disambiguation of the contents in the cells and the searching of the highest scoring entities which could represent the right concepts;
- at this point, a backward-update step kicks in to make an interpretation of the remaining data, guided by previously obtained results. This phase could modify the columns classification since there are new disambiguated entity content cells;

[6]http://www.programmableweb.com
[7]http://dublincore.org/documents/2001/04/12/usageguide/glossary.shtml
[8]http://dublincore.zsaorg/documents/profile-guidelines/



```
info:
  title: Bike sharing API
host: api.bikesharing.com
produces:
  - application/json
paths:
  /bikes:
    get:
      description: |
        The Bikes endpoint returns information about
available bikes at the nearest position.
      parameters:
        - name: lat
          in: query
          description: Latitude component of location.
          required: true
          type: number
          format: double
        - name: lng
          in: query
          description: Longitude component of location.
          required: true
          type: number
          format: double
      tags:
        - Bikes
      responses:
        200:
          description: An array of bikes
          schema:
            type: array
            items:
              $ref: '#/definitions/Bike'
```



```
info:
  title: Weather API
host: api.weather.com
produces:
  - application/json
paths:
  /weather:
    get:
    tags:
        - weather
      responses:
        200:
          description: An array of geographical area with
weather indication
          schema:
            type: array
            items:
              $ref: '#/definitions/GeoArea'
```

Figure 7.1: The APIs descriptions created following OAI specification

- finally, classifications and disambiguated entities are updated again with a mutually recursive pattern until they can be considered stabilised.

Another strong point in favour of TableMiner is the usage of predefined incremental inference with stopping algorithm, which does not require to analyse all the rows of a column, instead it stops when it feels confident, reducing considerably the computation time. Finally, TableMiner is adaptable to any knowledge bases.

We conceived our approach to use existing vocabularies (about 558) indexed in the Linked Open Vocabularies search engine[9]. The statistics presented in [116] and [134] shows that the most used vocabularies are not domain dependent and as such they may cover different topical categories such as media, government, publications, life sciences, geographic, cross-domain, user-generated content, and social networking. However, the above vocabularies may not provide all needed terms thus, we can rely on additional domain specific vocabularies to get a practical solution covering a large set of areas. They will be integrated with existing vocabularies to ensure practicability.

[9]http://lov.okfn.org/dataset/lov/

7.2 API Descriptions

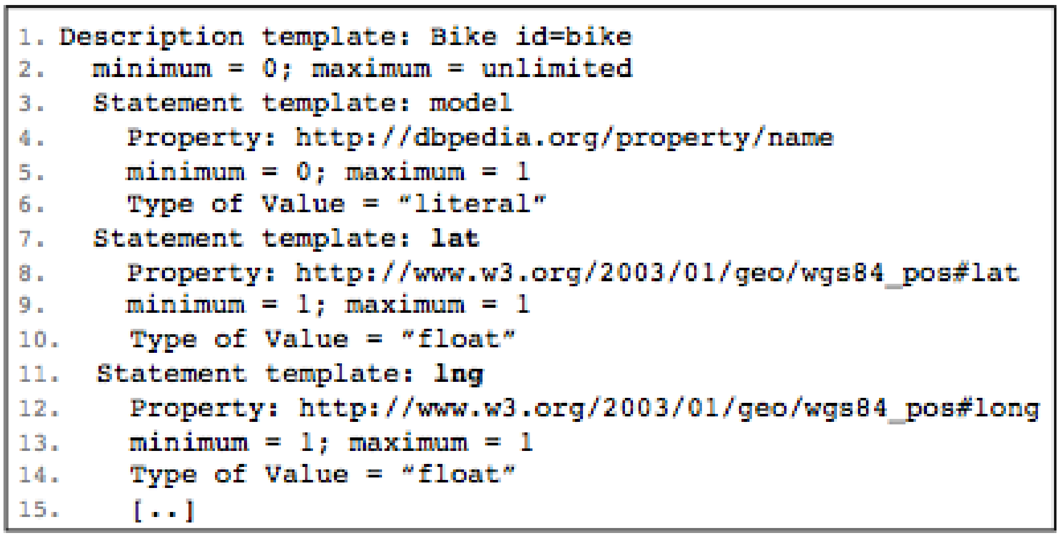
To analyse how we can enrich the existing REST description formats, let's discuss OAI descriptions that follow the Swagger format by providing an example involving *end-user developers*. Assume that a couple of tourists, John and Mary just arrived at one of the airports near Milan to visit the city. To move around they have different alternatives: (i) public transports, or sharing services for (ii) cars or (iii) bikes. Mary and John want to choose one of them according to preferences and/or context (e.g. weather, time, location, accessibility, etc.). Unfortunately, they have to invoke different information services to collect data before making an informed decision. Moreover, data are often not easily comparable or complete: for example, in the descriptions of bike sharing and weather APIs (Fig. 7.1) spatial references are in different formats and with different meaning (e.g., longitude/latitude versus area by points). Furthermore, most of the API descriptions are available only as HTML web pages, yet this is not adequate to support (semi) automatic comparison and composition of properties. We propose to extend OAI descriptions by mapping properties using DCAP to deliver API profiles, and defining a method for automatic extraction of concepts from shared vocabularies.

As for REST APIs, we should consider the use of HTTP OPTIONS method[10] to make REST APIs self-descriptive. Currently, the OPTIONS method is basically used to retrieve simple information about a resource, like the available HTTP methods that can be used in the communication with the specific API; however, since there is no standard response to an OPTIONS request, a full description could be returned in the response-body. With our approach API descriptions can be edited using the tool we developed, as explained in Section 7.4, and retrieved by invoking OPTIONS method. Other approaches that make the most of HTTP OPTIONS method are [125, 138].

7.2.1 API Profiles

By recalling the definition that we already discussed in Chapter 2 (section 2.6) "a profile is not to alter the semantics of the resource representation itself, but to allow clients to learn about additional semantics (constraints, conventions, extensions) that are associated with the resource representation, in addition to those defined by the media type and possibly other mechanisms" [144], it can be stated that API profile documents can offer a view of what is supplied by an API, and how clients and servers can expose features in a machine-readable format. The Dublin Core Metadata Initiative[11] (DCMI) released the DCAP format to describe profile metadata defining the constraints on how the RDF vocabularies are used to create profiles by linking properties. DCAP has been developed following the concept of Metadata Profile[12] that supports additional descriptive information about the contents of the response (e.g. useful indexing properties of the

[10]https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
[11]http://dublincore.org/
[12]https://www.w3.org/TR/html4/struct/global.html



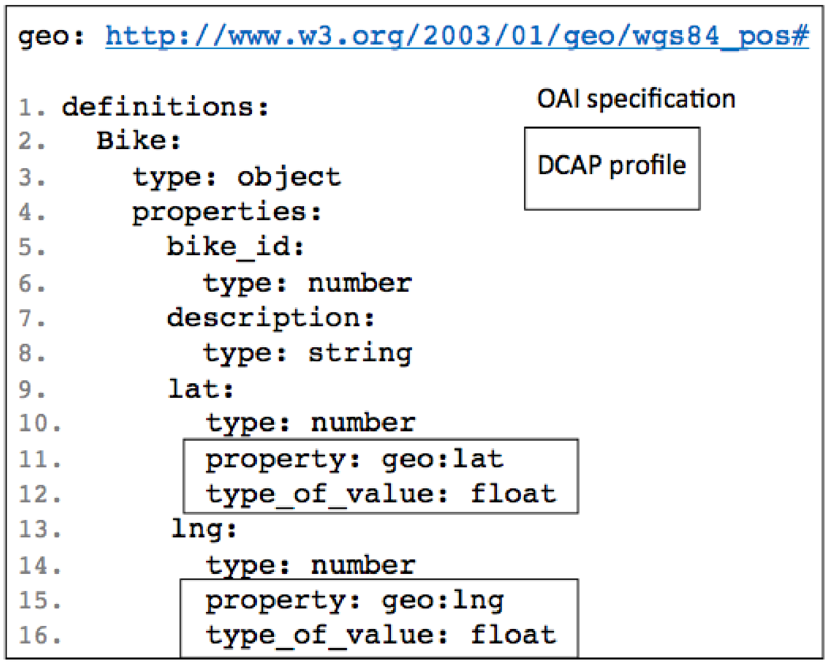
```
1.  Description template: Bike id=bike
2.    minimum = 0; maximum = unlimited
3.    Statement template: model
4.      Property: http://dbpedia.org/property/name
5.      minimum = 0; maximum = 1
6.      Type of Value = "literal"
7.    Statement template: lat
8.      Property: http://www.w3.org/2003/01/geo/wgs84_pos#lat
9.      minimum = 1; maximum = 1
10.     Type of Value = "float"
11.   Statement template: lng
12.     Property: http://www.w3.org/2003/01/geo/wgs84_pos#long
13.     minimum = 1; maximum = 1
14.     Type of Value = "float"
15.     [..]
```

Figure 7.2: An example DCAP profile

document, terms of use, etc.). For example in Fig. 7.2, line 8 and 12 link spatial data to concepts of latitude (lat) and longitude (long) from shared vocabularies. The approach is to enrich existing descriptions (Fig. 7.1) with such explicit references to shared vocabularies (Fig. 7.2) to facilitate automatic composition.

7.3 Semantic Annotations in API Descriptions

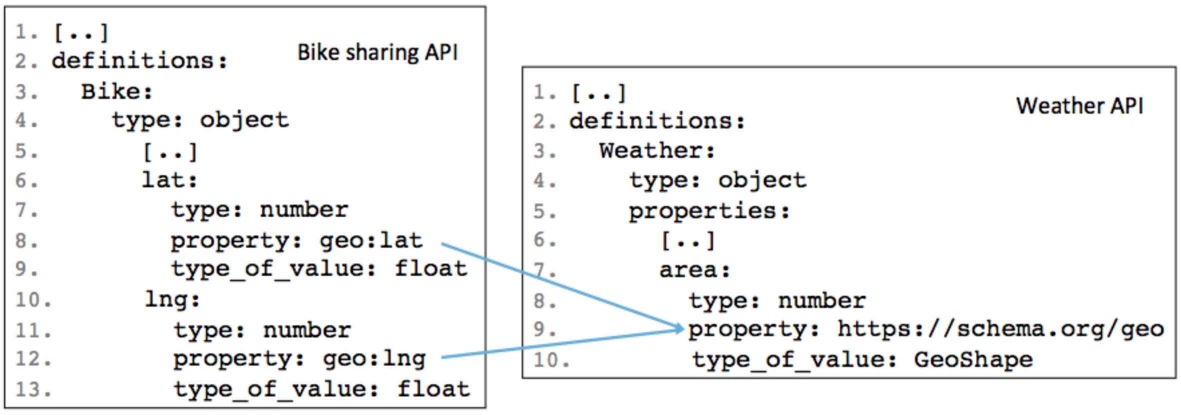
As standard description formats are missing, we propose to add semantic annotations in API descriptions through API profiles by linking properties to concepts in shared vocabularies. To show how the proposed approach works, let's go back to our example: Mary and John may save time and effort if we can provide all information related to useful services (e.g. public transport in Milan) in an aggregated way by composing descriptions. For example, if they can compare weather forecast information with available mobility services, they can make informed decisions like using bike sharing service in case of a sunny day, or a car sharing service in case of rain. To improve the descriptions in Fig. 7.1, we propose to add API profile information by means of DCAP specifications, as shown in Fig. 7.3 that specifies a definition of a bike. The resulting description consists of a comprehensive set of information that facilitate composition of responses. For example, in order to identify the best solution for Mary and John, car and bike sharing API responses can be enriched with contextual information such as weather forecast, traffic congestion, accessibility for disabled or elders, etc. In Fig. 7.4, the position of a bike is uniquely identified by concepts *geo:lat* and *geo:lng*. Similarly, information about the weather have been linked to concept *schema.org/geo*. In this way, it is possible to compare the responses of the two APIs and find the weather conditions in the area in which the bike is located. Similarly,



```
geo: http://www.w3.org/2003/01/geo/wgs84_pos#

definitions:
  Bike:
    type: object
    properties:
      bike_id:
        type: number
      description:
        type: string
      lat:
        type: number
        property: geo:lat
        type_of_value: float
      lng:
        type: number
        property: geo:lng
        type_of_value: float
```

Figure 7.3: Adding API profile using DCAP specification in OAI specification



```
[..]
definitions:
  Bike:
    type: object
      [..]
      lat:
        type: number
        property: geo:lat
        type_of_value: float
      lng:
        type: number
        property: geo:lng
        type_of_value: float
```

```
[..]
definitions:
  Weather:
    type: object
    properties:
      [..]
      area:
        type: number
        property: https://schema.org/geo
        type_of_value: GeoShape
```

Figure 7.4: Mapping of properties between responses of Bike sharing and Weather APIs

when the visitors want to use a car, they need to know if a parking place is available near their destination. They may use Google Places API by giving the value *parking* in the *types filter* for parking place searches. This API provides responses with the *address* property that shows geocoding results with a precise position. By correlating the semantics of properties of Weather API, *schema.org/geo*, and Google Places API, *schema.org/address*, the visitors can get required information to take a decision such as whether to book a open or closed parking place.

As we discussed in Chapter 4, an important issue is, often developers have to manually define the mappings between the information consumed and produced by Web APIs to shared

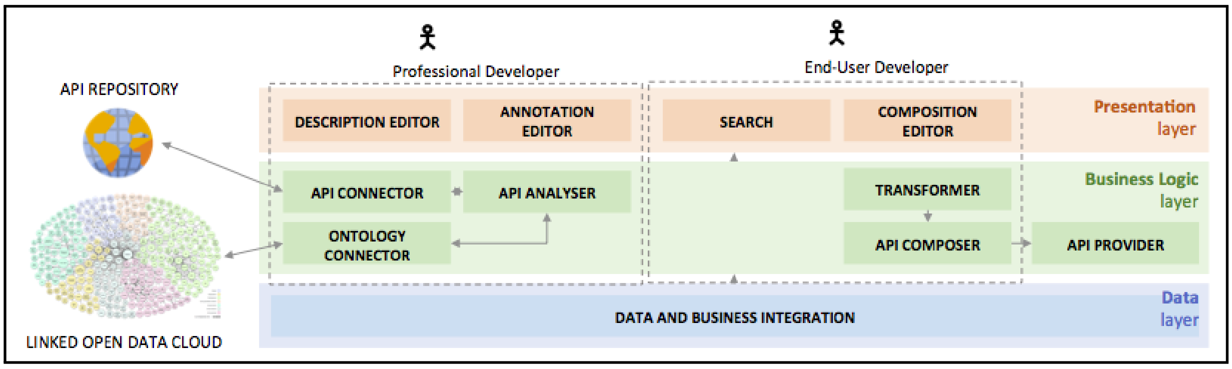


Figure 7.5: The system architecture

vocabularies. In the following section we define the architecture of the tool that can address the discussed issues presented in Chapter 4 about semantic annotation.

7.4 The Architecture - How the System Works

We have developed a system that target both *professional developers*, who have technical expertise in developing applications, and *end-user developers* that may not be familiar with the technologies discussed in the previous sections. Therefore, the aim is to create an abstraction layer to hide the technological complexity to the end-users and make the task of composing descriptions and services easier. The resulting system makes available a composition process through a REST API, which is provided by the *API Provider* component (Fig. 7.5).

The system architecture (Fig. 7.5) consists of three layers: *Presentation layer*, *Business logic layer* and *Data layer*. Both Presentation and Business Logic layers consist of components dedicated to different group of users: *Professional Developers* and *End-User Developers*.

The presentation layer dedicated to *professional developers* includes components that provide a user-friendly interface to enrich the descriptions of Web APIs, and manage semantic annotations. These tasks have been accomplished by extending the Swagger Editor, both for descriptions and annotations. In particular, the description process is semi-automatically managed by augmenting existing API descriptions, which can be retrieved from existing repositories (e.g. ProgrammableWeb), services registry, or by exploiting the HTTP OPTIONS method, as discussed in Section 7.2. Note that the OPTIONS method is also exploited by the system to support maintenance and evolution of descriptions already known by the system. If it is not possible to retrieve any initial description, the developer can insert a new API description manually using the Description Editor. These descriptions are represented in JSON or YAML format. For each resource, all relevant information such as available HTTP operations (e.g. GET, POST, PUT, DELETE, OPTIONS, HEAD, PATCH), the list of parameters for each operation, possible responses are collected. The process of creating a description is detailed in Algorithm 1.

Algorithm 1: Retrieve or create API description

```
Result: API description
1 if description is available then
2     retrieve description from existing repositories, services registries or via OPTIONS
      method;
3 else
4     create it manually using the Description Editor;
```

The Business Logic Layer allows developers to semi-automatically add semantic annotations to inputs and outputs by following the approach discussed in [150, 151]. The system automatically builds a table by putting properties in the header row, and filling up columns with API responses. These responses are collected dynamically by the *API Connector* component through multiple invocations of the involved APIs. The use of different input values allows the building of an accurate description of the APIs. In case of failure, the developer is asked to provide valid inputs to proceed. For the Bike sharing API example presented in Section 7.2, and Section 7.3, the header rows include input (e.g., “id_station”) and output (e.g., “station name”, “lat”, “long”, “free bikes”, “total slots”) properties, and the cell contents will be incrementally filled with data of each API invocation. So, the system is able to break up the response code (e.g. JSON, XML) in order to identify the output properties and their values.

In the Mary and John example, *lat*, *lng* and *area* are in the header row; values like “45.523", “9.219" and “[[45.524902, 9.216672],[45.526398, 9.218571], ...]”, which is an array of spatial points, fill up the column cells.

Algorithm 2 defines the process, which extends the one proposed by Karma [128] with the use of TableMiner technique to analyse the semantic properties of the resulting table. The TableMiner annotation technique is applied by the *API Analyzer* component that follows the steps described in Section 7.1 to set the meaning of properties by automatically linking them to concepts in the shared vocabularies. However, if the semi-automatic process fails, the developer is able to provide semantic annotation manually using the editor. Moreover, the system can support the developers by showing shared domain vocabularies and annotation alternatives.

Finally, the *Data and Business Integration* component stores the data to support the analysis of the APIs, and the produced descriptions and semantic annotations, to support the creation of an *ecosystem* of services. This *ecosystem* is an open set of services that could be automatically retrieved and linked each others to be able to follow the evolving user needs. The descriptions are retrievable by the OPTIONS method, as already discussed, to support use and evolution. In the context of this paper, we consider Mary and John as information seekers who want to know some information to facilitate their mobility. They are representative of generic users who wants to know more information about specific services (e.g., mobility) and related services. They are provided with descriptions to select and compose APIs according to patterns that have been

Algorithm 2: Create and add API profiles to API descriptions

```
Data: API description
Result: API description with API profile
1  Detect all resources' end-point;
2  foreach end-point do
       // collect data
3      repeat
4          generate input parameters following the API description;
5          if input parameters cannot be generated then
6              take input parameters from the user
7          invoke API with input parameters;
8          collect results;
9      until at least N results are collected;                /* default N=10 */
       // create tables
10     foreach results do
11         create a header row with API properties;
12         fill content-cells with values from inputs and responses;
   // add semantic annotations
13 foreach tables do
14     apply TableMiner technique;
15     show table to the user;
16     if table annotation is not complete then
17         show related vocabularies and/or alternatives to the user;
18         ask the user to manually add links;
19     if the user wants to review the annotations then
20         show related vocabularies to the user;
21         let the user confirm or modify the links;
22     insert API profile in API description;
```

pre-defined by professional developers.

The *Composition Editor* component is devoted to create compositions of services. The first step is to search for Web APIs that are already stored in the system by using the *Search* component. All relevant results and their possible combinations are loaded and showed in the interface to let the user select the APIs that match a set of given requirements. The second step is to create compositions either by directly linking the outputs and inputs of selected APIs, or by including *transformation* services that transform and make outputs compatible to inputs according to the semantic relations hold in the annotations. The *Transformer* component has the task of managing the set of transformation rules that make properties compatible. Composition patterns are then stored and provided to users such as Mary and John that have the task to populate such patterns with the services of interests.

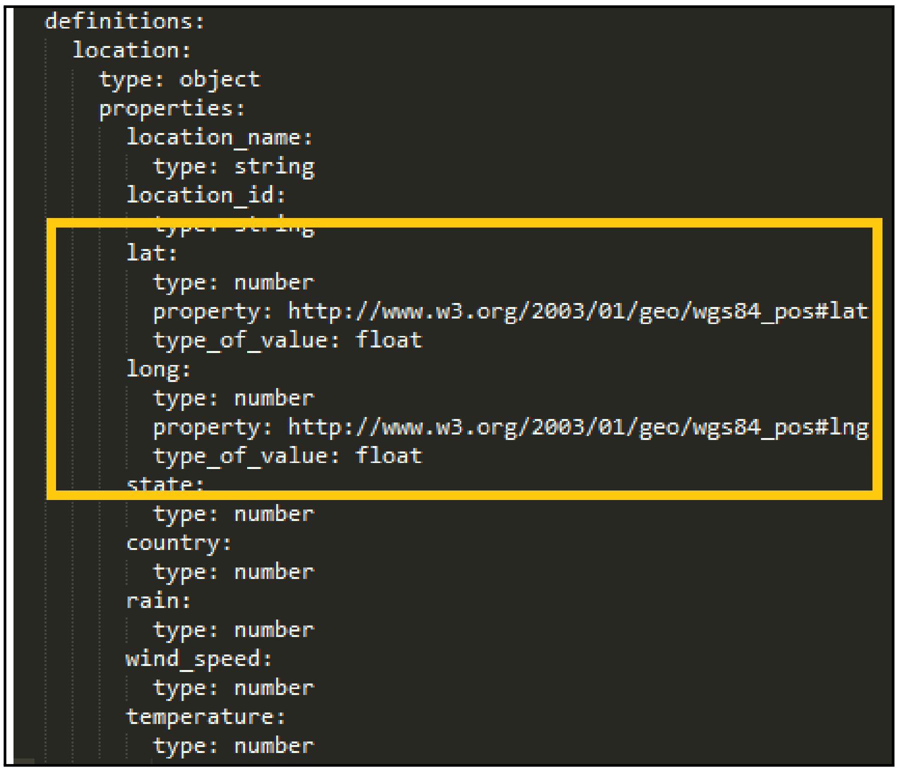
The Mary and John example can provide a general understanding of how these transformation rules work: after *lat* becomes *geo:lat*, and *lng* becomes *geo:lng* to build the augmented description (shown in Fig. 7.4), it is possible to identify which area includes the given pair of *geo:lat* and *geo:lng* values by applying the transformation rules. In such a way, it is possible to identify the weather conditions in the area in which a bike station is currently located. If the system cannot identify the appropriate rules to manage some annotated properties, the developer can insert new ones to enrich the system and ensure its evolution. In other words, the system provides end-users with synthetic information to accomplish a given task, anyway, if they are interested to see more details they can explore the process of transformation and composition. Moreover, if the user has the needed skills, he or she can contribute to the system evolution by adding transformation rules and/or composition patterns. One of the major advantage of the proposed system remains the separation of the annotation activities, which requires skills on semantic technologies, from the transformation and composition activities, which requires basic programming skills, or even no particular skills to just use the system as it is, like in the case of Mary and John.

7.5 (Semi) Automatic Composition: An Approach for Composing APIs through Semantic Annotations

In this section, we explain the composition process through the use of real APIs referring to the components of the system architecture (Fig. 7.5). As we described in the previous section, the professional developers are responsible for creating the enriched description where the functionalities are provided by the API analyzer, API connector and API analyzer components. They use the *Description Editor* for creating or editing the descriptions (Fig. 7.6) and add semantic annotations using the *Annotation Editor* (Fig. 7.7). Moreover, they create the transformation rules or use external APIs to make the input-output compatible (Fig. 7.8). In the composition scenario showed in Fig. 7.9, the user wants to compose Car2Go API with Forecast Weather API. As the system uses pipeline structure, the outputs of Car2Go API (e.g. coordinates (lat/long)) needs to be compatible with the inputs of the Forecast Weather API (e.g. city name), but as these output-input is not compatible, the external API named as Transformer API is called by the Transformer component that finds the city name according to the given latitude and longitude. This is done automatically by using the semantic information presented in the enriched descriptions. The professional developers also provide the composite APIs based on the compatibility of the involved Web APIs.

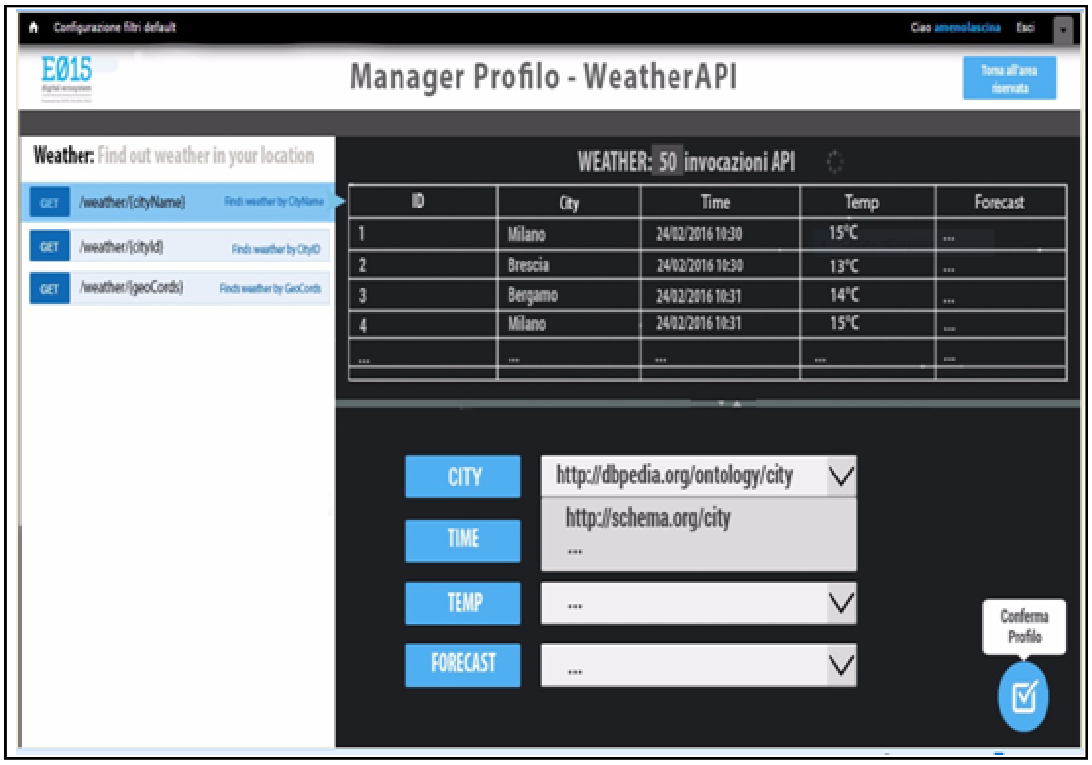
7.6 Summary

As we discussed in Chapter 3 (Problem Definition) and Chapter 4 (Related work), Web APIs are associated with textual descriptions that are not understandable by machines and cannot be composed (semi) automatically. There exist approaches, including WADL, WSMO Lite, Resource-



```
definitions:
  location:
    type: object
    properties:
      location_name:
        type: string
      location_id:
        type: string
      lat:
        type: number
        property: http://www.w3.org/2003/01/geo/wgs84_pos#lat
        type_of_value: float
      long:
        type: number
        property: http://www.w3.org/2003/01/geo/wgs84_pos#lng
        type_of_value: float
      state:
        type: number
      country:
        type: number
      rain:
        type: number
      wind_speed:
        type: number
      temperature:
        type: number
```

Figure 7.6: Enriched description of Weather forecast API



Configurazione filtri default

Ciao ammolascina Esci

Manager Profilo - WeatherAPI

Weather: Find out weather in your location

GET /weather/{cityName} Finds weather by CityName

GET /weather/{cityId} Finds weather by CityID

GET /weather/{geoCords} Finds weather by GeoCords

WEATHER: 50 invocazioni API

ID	City	Time	Temp	Forecast
1	Milano	24/02/2016 10:30	15°C	...
2	Brescia	24/02/2016 10:30	13°C	...
3	Bergamo	24/02/2016 10:31	14°C	...
4	Milano	24/02/2016 10:31	15°C	...
...	...	...	...	...

CITY http://dbpedia.org/ontology/city
http://schema.org/city
...

TIME

TEMP ...

FORECAST ...

Figure 7.7: Editor for professional developers to add semantic annotation

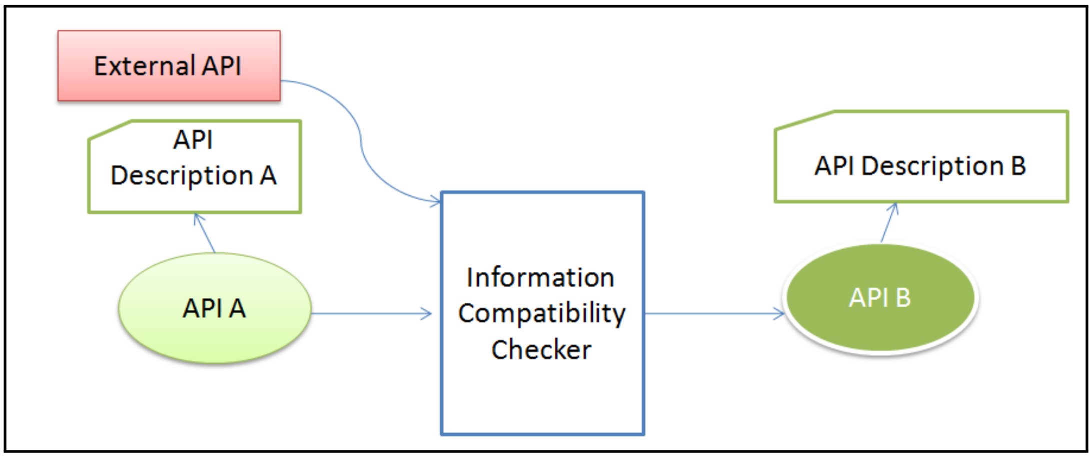


Figure 7.8: The compatibility checker

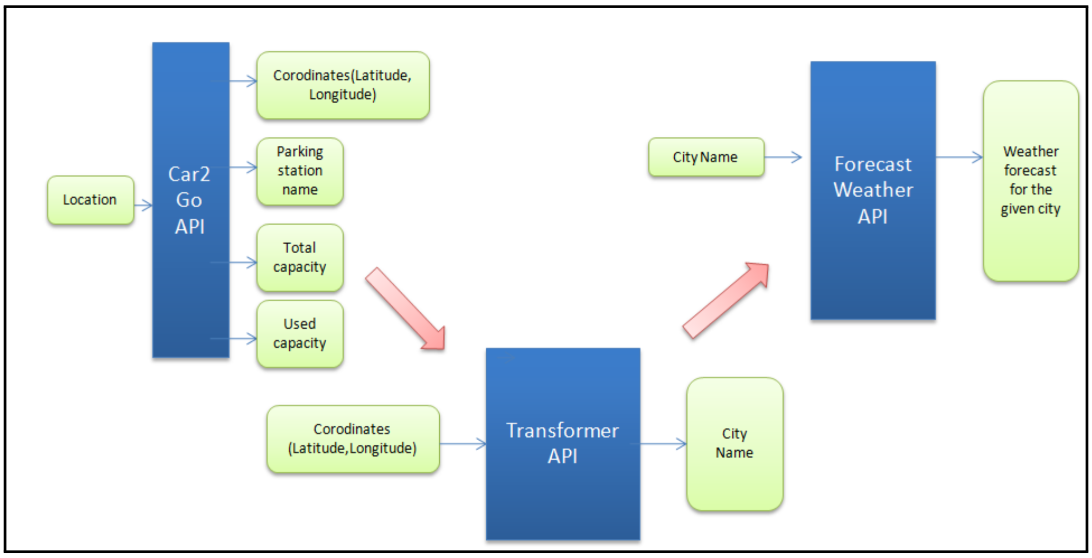


Figure 7.9: Input/Output compatibility matching using Transformer API

Oriented Service Model (ROSM) and RESTdesc, that provide rich semantic descriptions, but they are not widely adopted because of the required expertise in Semantic Web Languages (e.g. RDF, SPARQL, N3) as well as in-depth domain knowledge [136]. Although machine-readable descriptions (e.g. MicroWSMO, Minimal Service Model, SA-REST) have been introduced to support additional semantic information, tools for creating automatic or semi-automatic semantic annotations are missing. In this chapter, we describe how the semantic annotations can be (semi) automatically added to the API descriptions. To accomplish this, we made an extension to Open API Initiative (OAI) Specification following the Dublin Core Application Profile (DCAP) guidelines. We also describe the composition process using the enriched descriptions that target the users who are technology experts and professional developers that can understand the involved concepts and drive the semi-automatic tool we developed. Our next work is to present high-level concepts that target specific requirements (e.g. common needs and requirements of specific groups of users), and are understandable by generic users. Such concepts are implemented in visual interfaces that can support actual user evaluation tests. In the next chapter, we present our approach for bridging the gap between the technology constraints and end-user' requirements.

Chapter 8

Bridging the Gap between End-users' Requirements and Technology Constraints

This chapter describes the approach that bridges the gap between the constraints of the used technologies and the requirements of the end-users. As one of our driving goals is to accommodate all groups of end-users in Web API composition environment, we design and develop a prototype considering the requirements we gathered in chapter 5 by analyzing the context of older domain. This approach requires to design an easy to use API composition tool that can be adapted by all groups of end-users.

8.1 Mismatch between Requirements and Implementations

A major problem in Information Systems (IS) development is misalignment between needed functionality and the functionality offered in the developed IS. This could be stated as being a continuous challenge - independent on if it is internal development or if it is procurement of standard software package [60]. This problem could be described as the distance between what end-users want to have support for in the business processes, and what the Information Systems de facto gives support for. There are definitely a lot of different reasons for why this is the case. But, it can be stated that one important consideration is the difficulties of translating and transferring business requirements from identification to specification, and further into implementation. In systems engineering and requirements engineering, a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors [105]. They are contrasted with functional requirements that define specific behavior or functions. There have been attempts both from research and practice to classify and categorize requirements, resulting in classification schemes that distinguish between functional and non-functional requirements. By analyzing the existing literature [56], it can be stated that the scope of what requirements are have broadened over time. It is important to identify the business requirements and also focus on how the developed system will execute the wanted requirements.

A high-level distinction between functional and non-functional requirements has been identified by the existing software engineering approaches. However, there are criticism in defining software specification [92], such as Unified Modeling Language (UML) techniques are not suitable for translation of business models into software models due to the fact that technical methodology does not take organizational aspects into account such as dynamic internal political agendas, conflicting interests and interpretations among the involved stakeholders. Nevertheless, and returning to the high-level separation [85], functional requirements represent the type of operations that connects the user and problem domain with the representation of the problem domain and non-functional requirements represent requirements beyond the above mentioned; for example usability, computing efficiency, reliability, scalability, reusability, portability, etc.

Non-functional requirements are often called "quality attributes" of a system [126]. Other terms for non-functional requirements are "qualities", "quality goals", "quality of service requirements", "constraints" and "non-behavioral requirements". Qualities that represents non-functional requirements can be divided into two main categories:

- Execution qualities, such as security and usability, which are observable at run time.
- Evolution qualities, such as testability, maintainability, extensibility and scalability, which are embodied in the static structure of the software system.

In our work, we emphasize on usability that resides in the execution quality and are observable at run time.

Usually Web applications provide both functional and non-functional information. For example, applications like ATM[1] transport service provides information on routes and stops, and information on accessibility (e.g., elevators at Metro stations) and fare zones. Also there are static and dynamic information that can be retrieved. For example, the ATM transport service shows static time table for buses and it also shows dynamic information like when the bus will come to the specified bus stop. Most of the time, developers only focus on functional parts of their applications due to its importance. Although non-functional parts are much more difficult to design and to implement than business ones, these parts should be considered to fulfil end-users' needs. We consider these aspects and include additional information that can fulfil non-functional requirements by providing dynamic information such as an elevator is working or not at the specified station or stoppage. In the next sections we describe the approaches for supporting user driven API composition through technological solutions and usability features.

[1]http://www.atm.it/en/ViaggiaConNoi/Pages/ATMMobile.aspx

8.2 Supports for User-driven Composition of Web APIs: Technology Stacks for Bridging the Gap

To be consistent with a user driven approach we consider to enable the end-users to execute dynamic composition. As showed in figure 8.1, the actions taken by the professional developers are:

- they discovers or develops Web APIs,
- they discovers or creates descriptions,
- they annotates Web API descriptions by correlating properties with the concepts of shared vocabularies to provide enriched descriptions, and
- they creates the composite Web APIs, by defining the composition patterns and adding enriched descriptions.

The following data are deposited to the Web API repository:

- Single Web APIs,
- Enriched Web API Descriptions, and
- Composite Web APIs

For the end-user developers, the interface of the Composition Editor is provided in a way where the Web APIs are presented as services. These services can be a single service (i.e weather service or car sharing service) or a composite service (i.e. a service that contains other services). The end-user can select any single service or a composite service. When they selects a composite service they can modify it by adding services. The result is that end-users are acting as developers as they modify a service by working with a presented set of available services, and save the result as a new service, which become available for future use.

The *Composition Editor* dedicated for end-user developers has been constructed by considering:

- a set of Web APIs with enriched descriptions (e.g. with semantic annotation),
- a single step approach for selecting Web APIs directly,
- a series structure or pipeline approach where an API needs data from previous API in the composition process.
- a set of Transformer APIs to make the inputs and outputs compatible, and
- a set of Web APIs for providing additional information such as dynamic accessible information (e.g. elevator is working or not in a specific metro station).

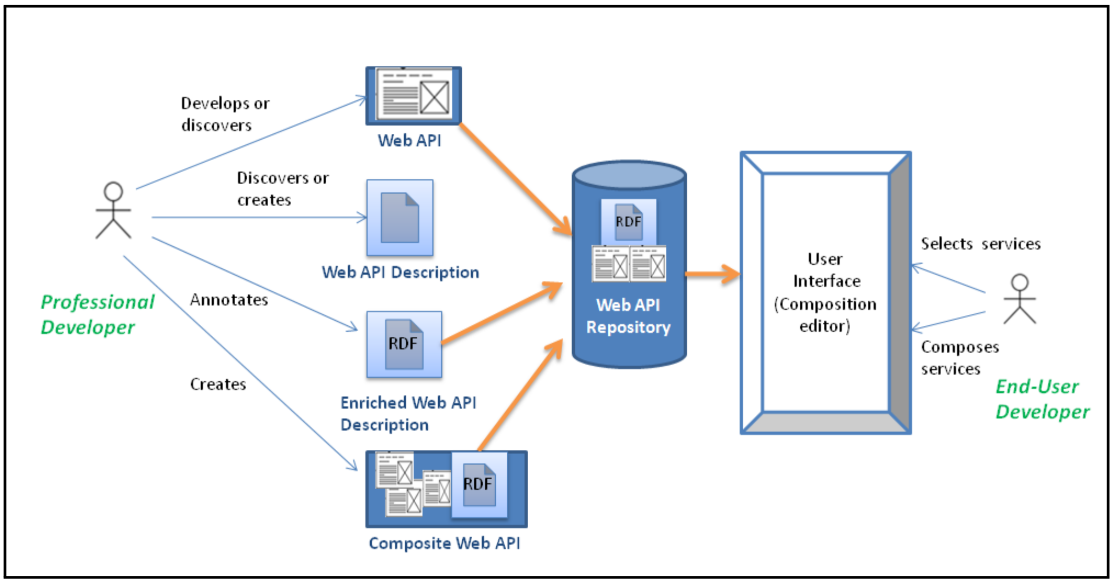


Figure 8.1: User-driven composition

Several literature on end-user development approach [4, 40] emphasize on avoiding technical terms to make the systems or applications more applicable for a wide range of end-users. This is because, end-users who are not expert users of Information Systems or applications, can get confused with the technical terms. Moreover, they just want to integrate information by selecting services, instead of knowing the technological details. Moreover, they want an easy to use application with understandable interfaces to complete their required tasks. In the next section we describe the supports that we have provided for end-user developers to facilitate usable interactions.

8.3 Supports for End-User Developers (Including Elders): Usability Features for Bridging the Gap

In interaction design, prototypes are commonly used as shared representations of design ideas to bridge the gap between the involved communities. It may, however, be difficult for designers, developers, and providers to know which prototyping technique to use to express the design ideas. Thus the choice of prototyping technique should be based on the features resulted from the combination of the involved people and the desired focus of the tasks [61]. In the following subsections we describe our considerations during low-fidelity design and prototype implementation.

8.3.1 Low Fidelity Prototype Design

To support the end-users in the Web API composition environment, at first we adopted the low fidelity prototype method to design our expected tool (see Fig. 8.2 and 8.3) that can facilitate end-user developers. At this point we recall all the design requirements elicited from the context analysis of elder domain described in chapter 5, as listed in the following:

- Privacy
- Set of Controls to support elders (visual magnifier, sound control etc.)
- Clear structure of tasks
  - Page function unity (one page-one task)
  - Place confirmations every possible time
  - Distinct feedback from each actions
- Reduction of complexity
  - Avoid use of complex interactions
- Operable user interface and navigation
  - Clear navigation and location (search option, site map, descriptions of titles of pages/ results of search, indication of current location)
  - Clear purpose of a link
  - Sufficient time
- Understandable Metaphors and Icons
- Perceivable information and user interface (text size, color contrast, text style etc.)
- Understandable information and user interface (consistent navigation, instructions and input assistance, understandable language, error prevention and recovery for form)

After analyzing this prototype by in-lab experiments and getting opinions from 3 expert designers, we found that some changes need to be done while we are considering also elders as end-users, such as a limited set of controls should be at the bottom and use of list-boxes might be harder to use due to the number of clicks required. The check box/radio-button form allows a direct selection from the choices, whereas the form with list-boxes are provided with a small down arrow that needs to be clicked to see the available options. They also suggested to remove volume control as it is not related to our service composition scenarios. Also they emphasize on making the interfaces simpler with clear instructions and reducing the number of clicks. Thus, we consider these features during the implementation of our prototypical solution.

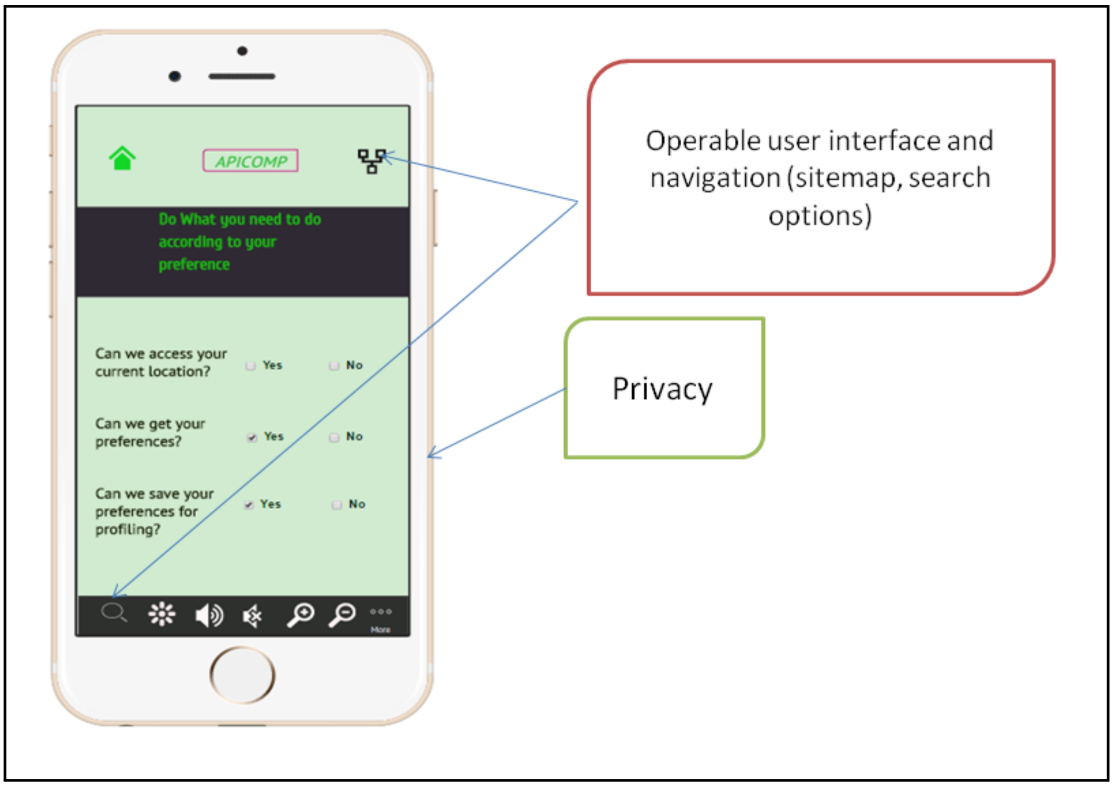

APICOMP
Do What you need to do according to your preference
Can we access your current location? Yes No
Can we get your preferences? Yes No
Can we save your preferences for profiling? Yes No
More
Operable user interface and navigation (sitemap, search options)
Privacy

Figure 8.2: Low-fidelity prototype design with operable user interface.

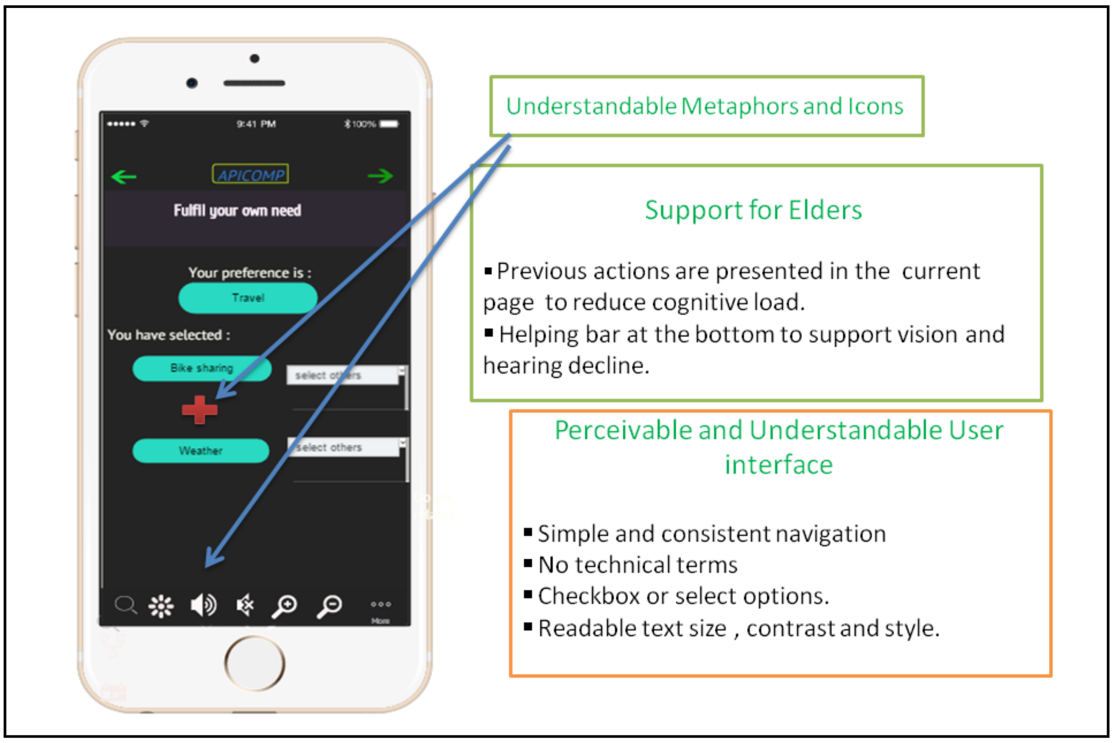

9:41 PM
100%
APICOMP
Fulfil your own need
Your preference is :
Travel
You have selected :
Bike sharing
select others
Weather
select others
More
Understandable Metaphors and Icons
Support for Elders
▪ Previous actions are presented in the current page to reduce cognitive load.
▪ Helping bar at the bottom to support vision and hearing decline.
Perceivable and Understandable User interface
▪ Simple and consistent navigation
▪ No technical terms
▪ Checkbox or select options.
▪ Readable text size , contrast and style.

Figure 8.3: Low-fidelity prototype design considering all groups of end-users (including elders).

8.3.2 Prototype Implementation

We implemented our prototype using JAVA Script, JQuery, HTML, CSS, PHP and MySQL. The information at the front-end is presented using texts and metaphors. The purpose of the interface metaphor is to give the user instantaneous knowledge about how to interact with the user interface. By definition, an Interface metaphor is a set of user interface visuals, actions and procedures that exploit specific knowledge that users already have of other domains [118]. They are designed to be similar to physical entities but also have their own properties (e.g., desktop metaphor and web portals). They can be based on an activity, an object, or a combination of both and work with users' familiar knowledge to help them understand the "unfamiliar" objects or actions, and placed in the terms so the user may better understand. An example of an interface metaphor is the folders and the file cabinet representation of the file system of an operating system. Another example is the tree view representation of a file system, as in a file manager, that helps a user to use it, given some previous knowledge of recursive structures.

8.3.3 Interface Controls and Operations

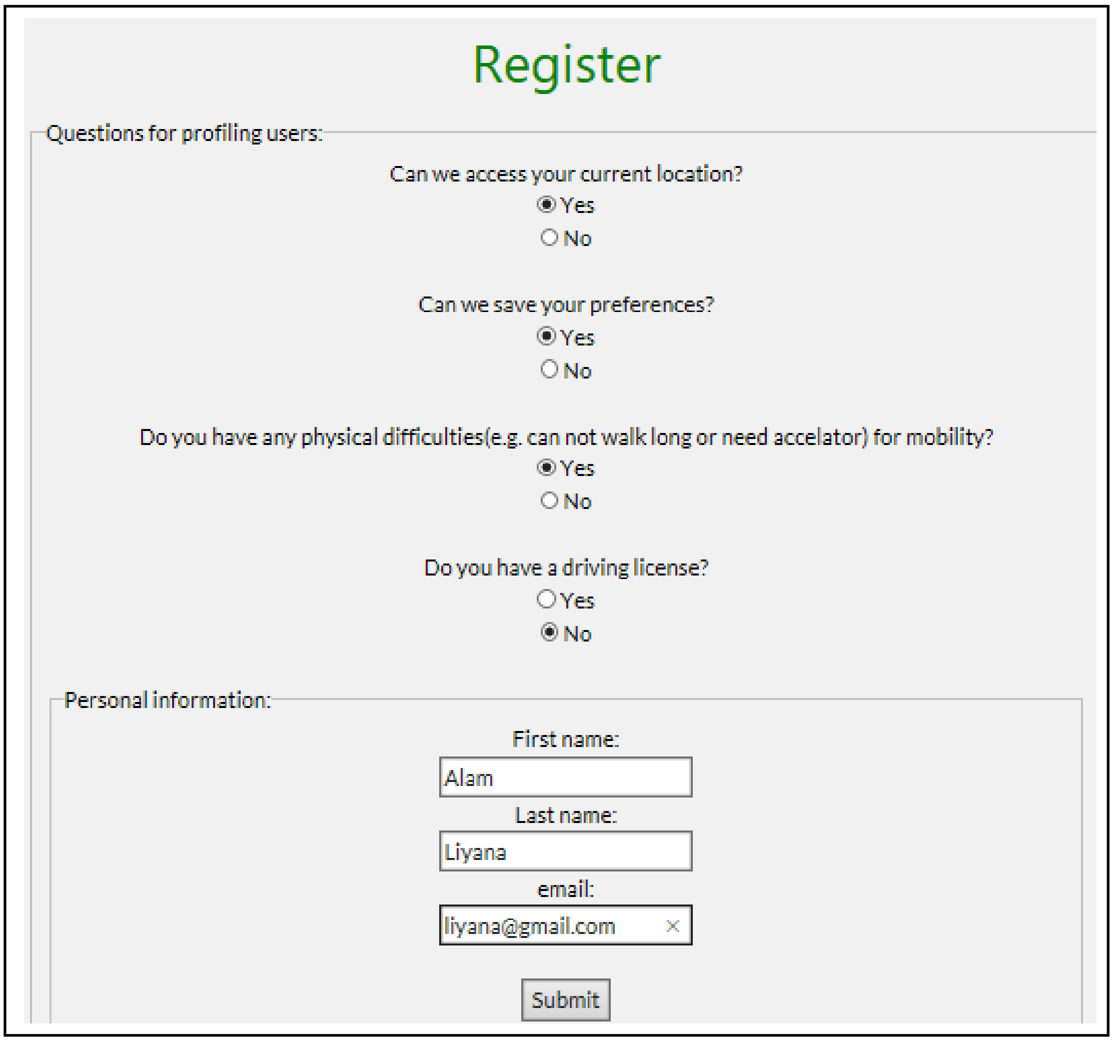
For the sake of simplicity we implemented the prototype considering one of the most relevant scenario, that is "Mobility in a city" and named it as "Personalized Mobility Service". The implementation of the prototype followed a basic principle: to provide the functionalities of the composition environment through an easy interaction style. To make it more adaptable to all groups of end-users, we considered all the interface design requirements gathered in chapter 5.

8.3.3.1 Registration

When a user enters in to the portal of the Personalized Mobility Service for the first time, he/she has to register with the required information through which user profiles is created. To complete the registration he has to fill a user profile form with the information to ensure the privacy of the users and accessibility related questions (Fig. 8.4). He also has to fill some mandatory fields like first name, last name and email address. After successful submission of the form the user can see a confirmation message on the screen (Fig. 8.5). Also a password is sent to the user through an email that can be changed by the user at any time. If the user has already been registered, he can log into the system with the authorized user name and password.

8.3.3.2 Navigation

After log in to the service the user can see a navigation bar (Fig. 8.6) on the screen to access the service functionalities such as Home, Search, My Service and Help Button that shows two links: Site Map and Text Resize options. These links are provided for readily accessible information such as a list of available services (Fig. 8.7), details of the services (Fig. 8.8), already combined services by the user (Fig. 8.13) and some help options required by the users. For the main functionalities



Register

Questions for profiling users:

Can we access your current location?
[✓ Yes] No

Can we save your preferences?
[✓ Yes] No

Do you have any physical difficulties(e.g. can not walk long or need accelator) for mobility?
[✓ Yes] No

Do you have a driving license?
Yes [✓ No]

Personal information:

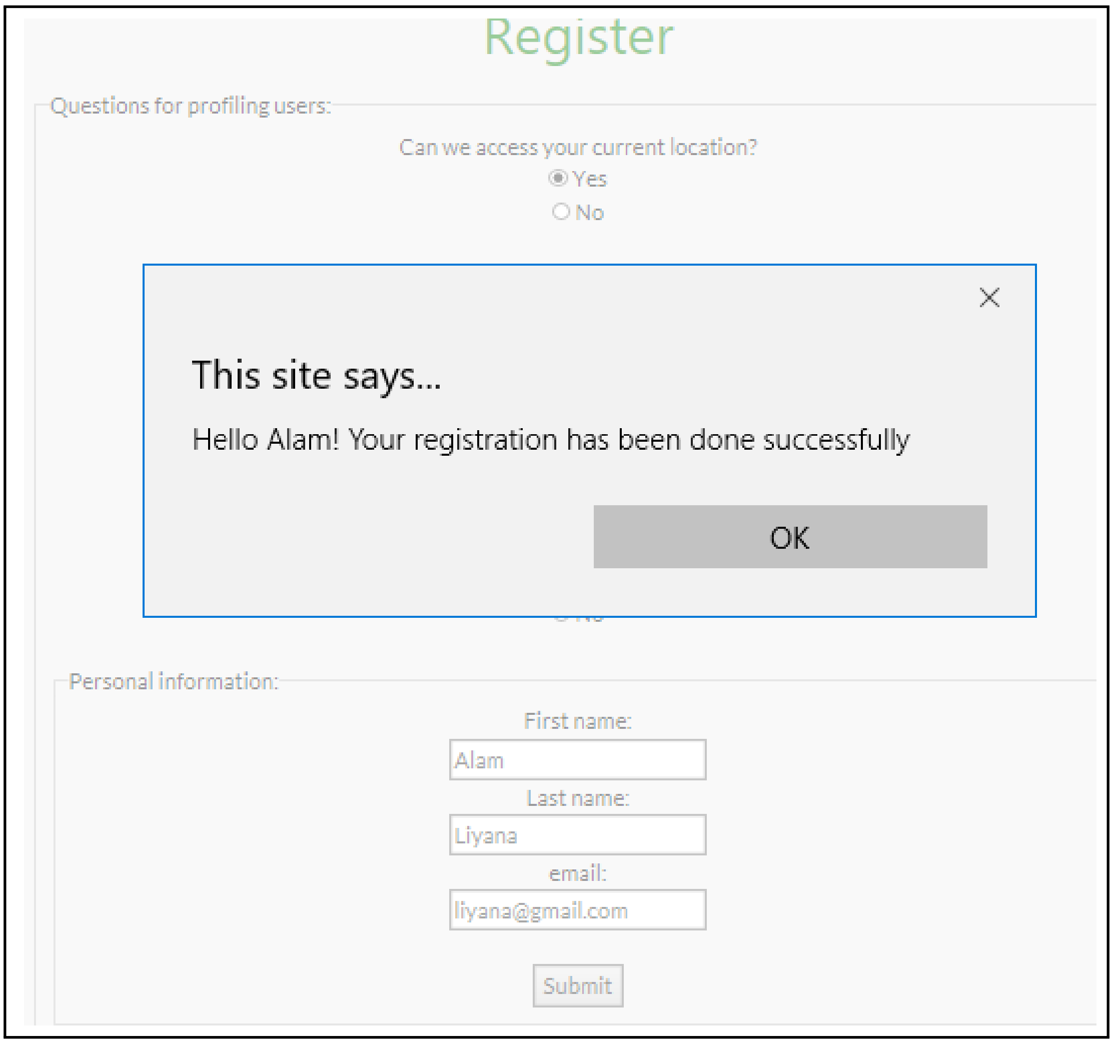
First name:	Alam
Last name:	Liyana
email:	liyana@gmail.com

Figure 8.4: Registration page

such as using services and composing services the system guides the users through links on current pages.

8.3.4 The Invocation of Services

To go somewhere, the user has to mention the name of the starting place and destination to specify the route. The resulting page shows the possible options for going to the destination place from the starting point such as public transportation services, bike sharing services or car sharing services (Fig. 8.9). He can select any one of the showed options. For example, he can select bike sharing service by clicking the “Bike Sharing" option. At this point the user gets the information provided by the bike sharing API resides at the back-end (Fig.8.10). By assuming the fact that the users are not interested in the available bike sharing services, we implemented the system in a way that invokes the top ranked API to show the output. In case, the user wants



Register

Questions for profiling users:

Can we access your current location?

(●) Yes

( ) No

This site says...

Hello Alam! Your registration has been done successfully

OK

Personal information:

First name: Alam

Last name: Liyana

email: liyana@gmail.com

Submit

Figure 8.5: Confirmation feedback

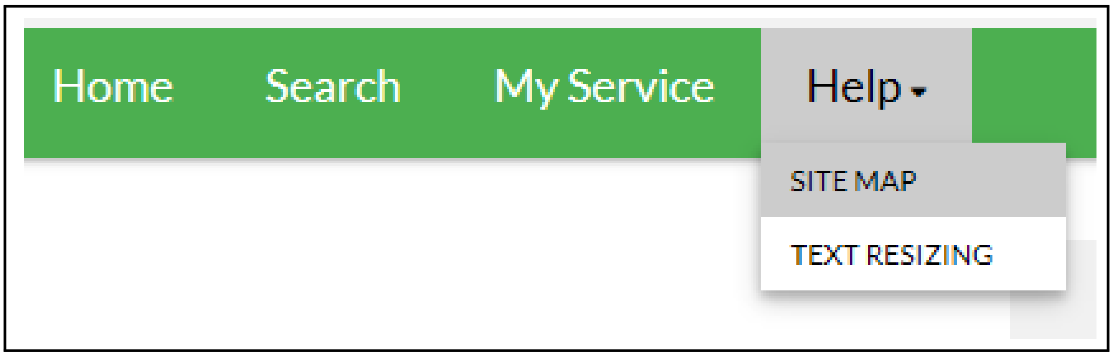


Figure 8.6: Navigation bar

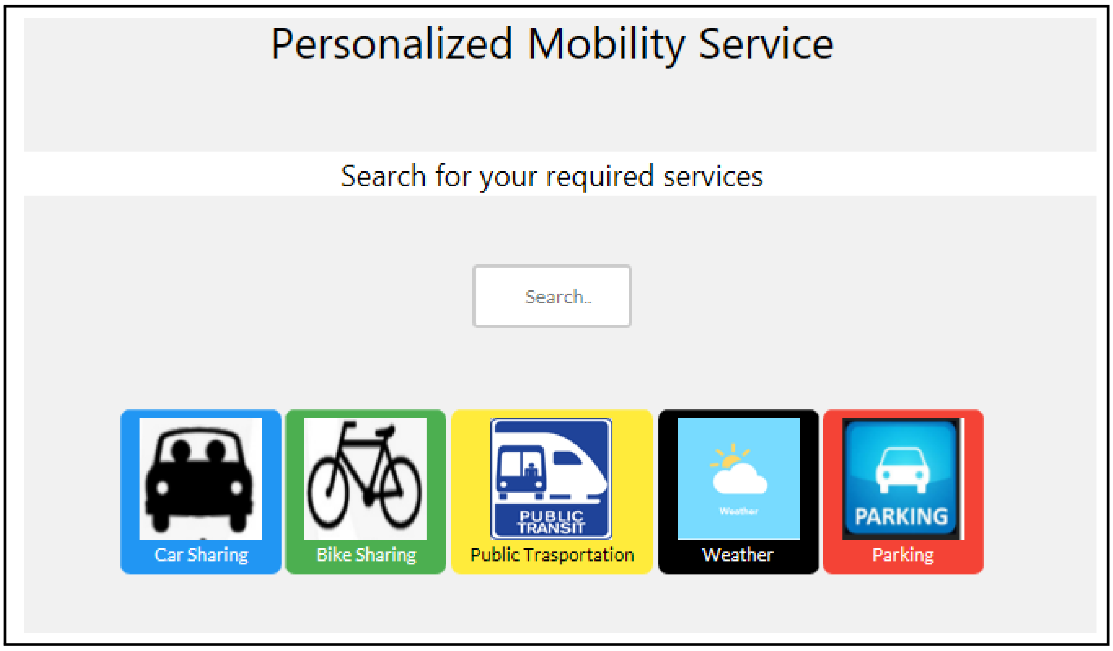


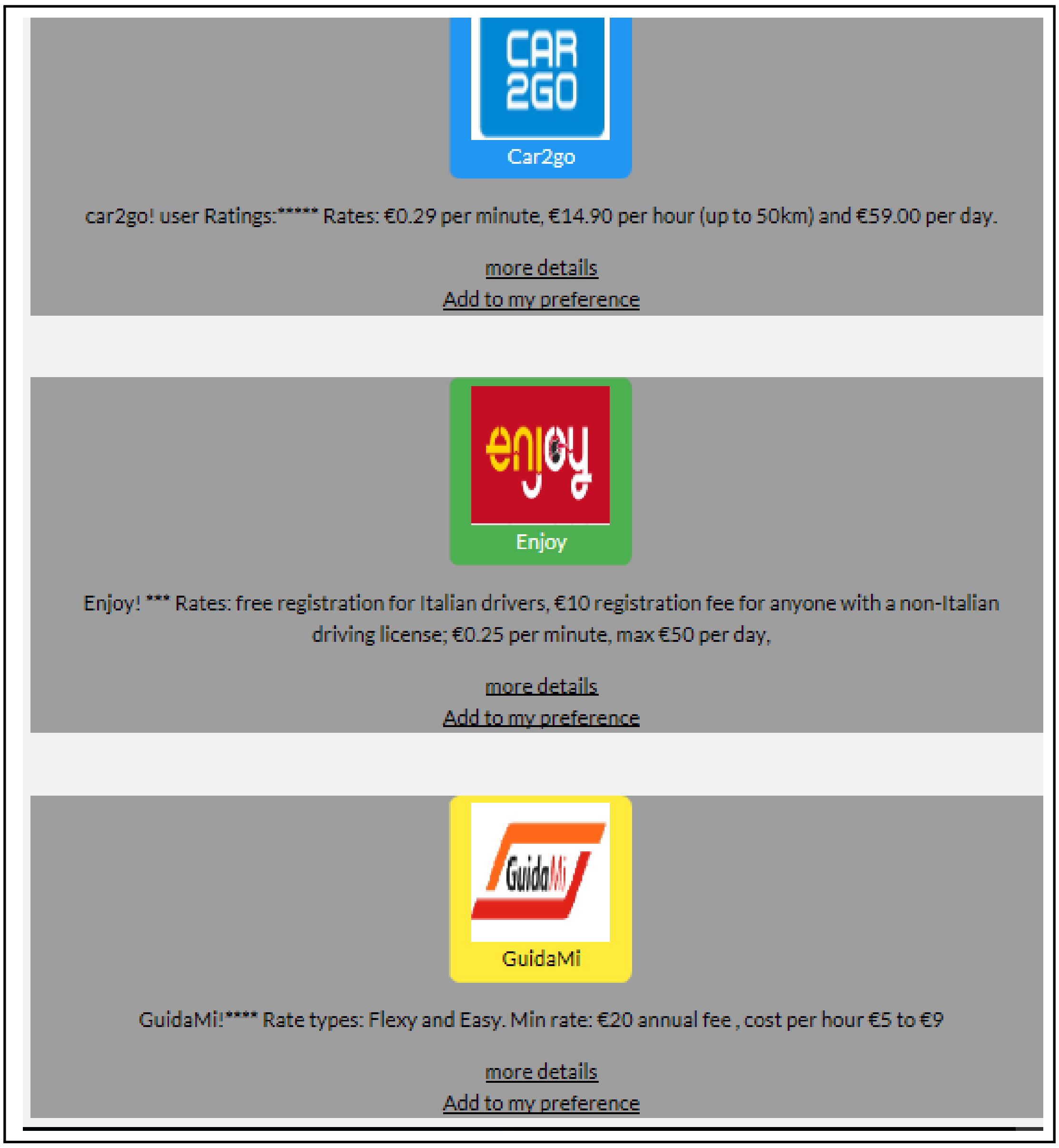
Figure 8.7: Search

to choose among available services, he can go the service details page and select his preferred service.

8.3.5 The Composition of Services

This prototype provides composition option by guiding the users through a link "Click for Related Information" (see Fig:8.11). By clicking this link the user can get the possible options for integrating information using available services. This availability depends on the composition patterns provided by the professional developers. At this page, he can choose to get related information such as weather information of the specified place or parking at the destination place. For example, if he selects to get weather information the weather API is invoked at the back-end and composition is executed between the selected bike sharing API and Weather API (Fig. 8.12) through the use of semantic annotation presented in the enriched API descriptions as described in chapter 7. As a result, the user gets the integrated information that is helpful for taking decision such as in case of rain or heavy wind it is not possible to use a bike resides at that particular station. By clicking the "Add to My Service" the composed service is saved in the "My Service" that can be used as a preferred combined service for future use (Fig:8.13).

Similarly, other mobility options can be chosen by the user. For example, the user selects the public transport option that shows the information for going from the starting point to the destination point. Although the invoked public transport API provides some static information such as presence of an escalator or elevator at the specified station, the dynamic information (e.g. the elevator is working or not) is missing. The user may wants to get this related information by



car2go! user Ratings:***** Rates: €0.29 per minute, €14.90 per hour (up to 50km) and €59.00 per day.

Enjoy! *** Rates: free registration for Italian drivers, €10 registration fee for anyone with a non-Italian driving license; €0.25 per minute, max €50 per day,

GuidaMi!**** Rate types: Flexy and Easy. Min rate: €20 annual fee , cost per hour €5 to €9

Figure 8.8: Car Sharing service details

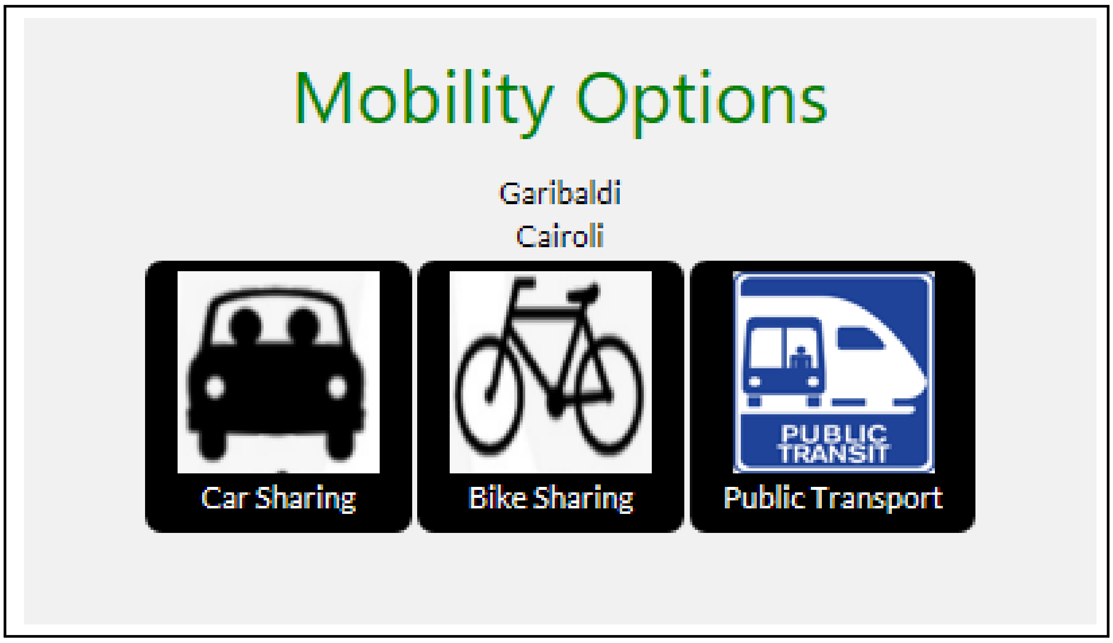
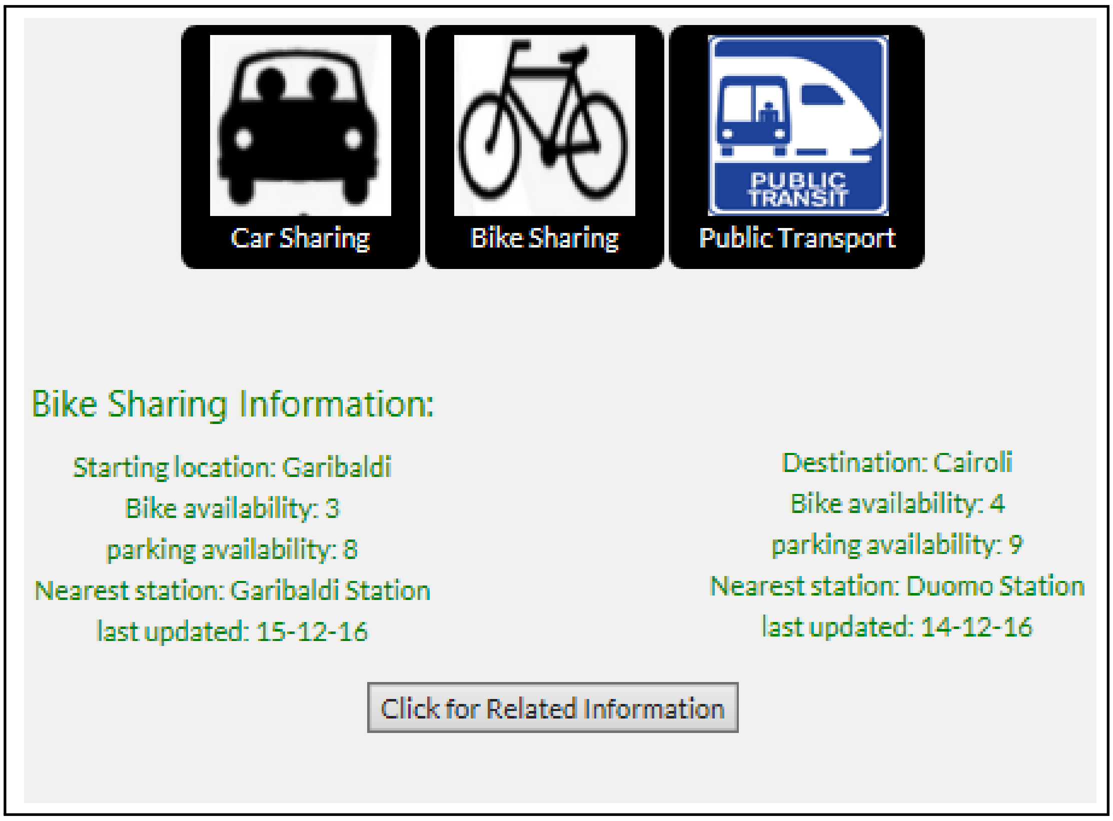


Figure 8.9: Mobility options

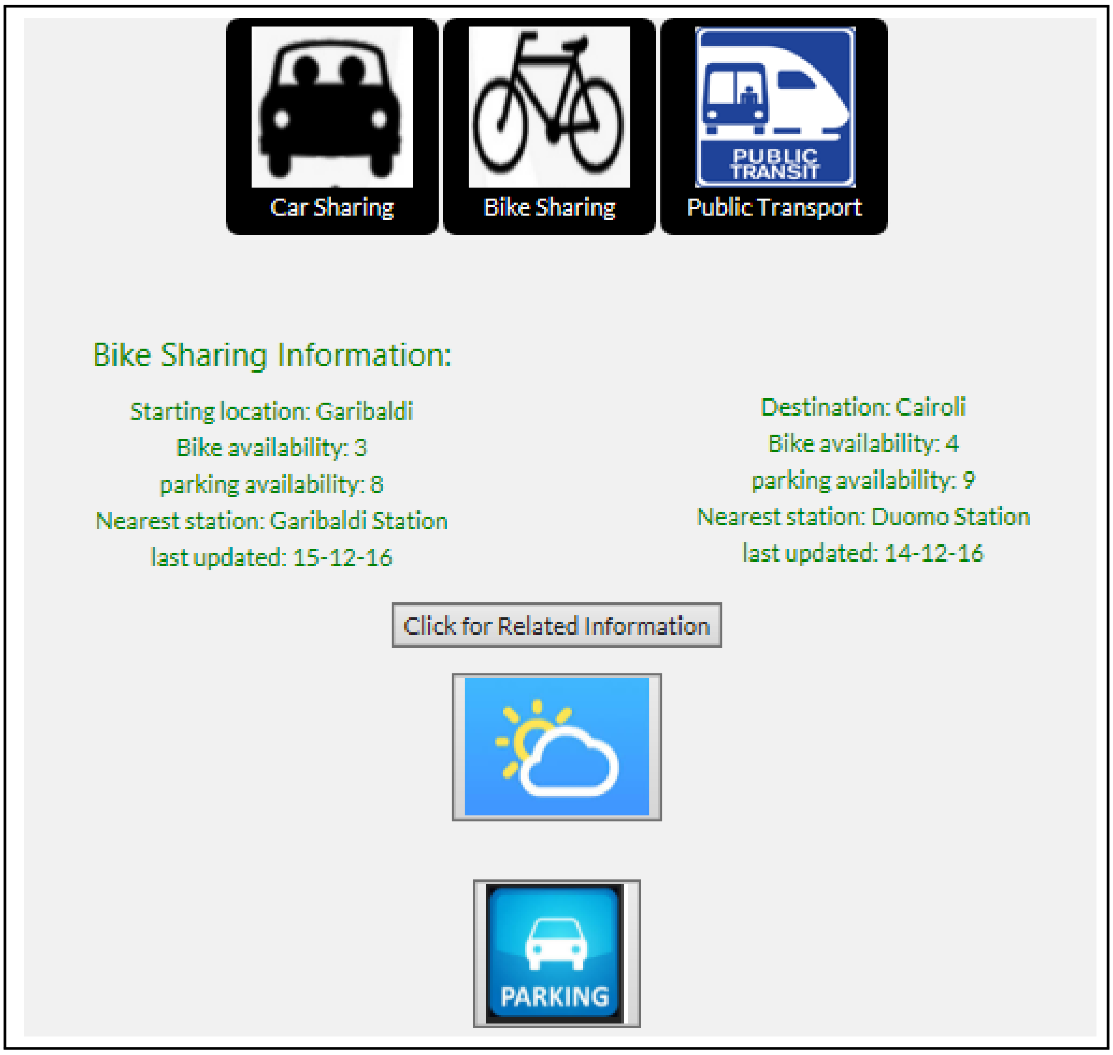


Bike Sharing Information:

Starting location: Garibaldi
Bike availability: 3
parking availability: 8
Nearest station: Garibaldi Station
last updated: 15-12-16

Destination: Cairoli
Bike availability: 4
parking availability: 9
Nearest station: Duomo Station
last updated: 14-12-16

Figure 8.10: Output from Bike Sharing service



Bike Sharing Information:

Starting location: Garibaldi
Bike availability: 3
parking availability: 8
Nearest station: Garibaldi Station
last updated: 15-12-16

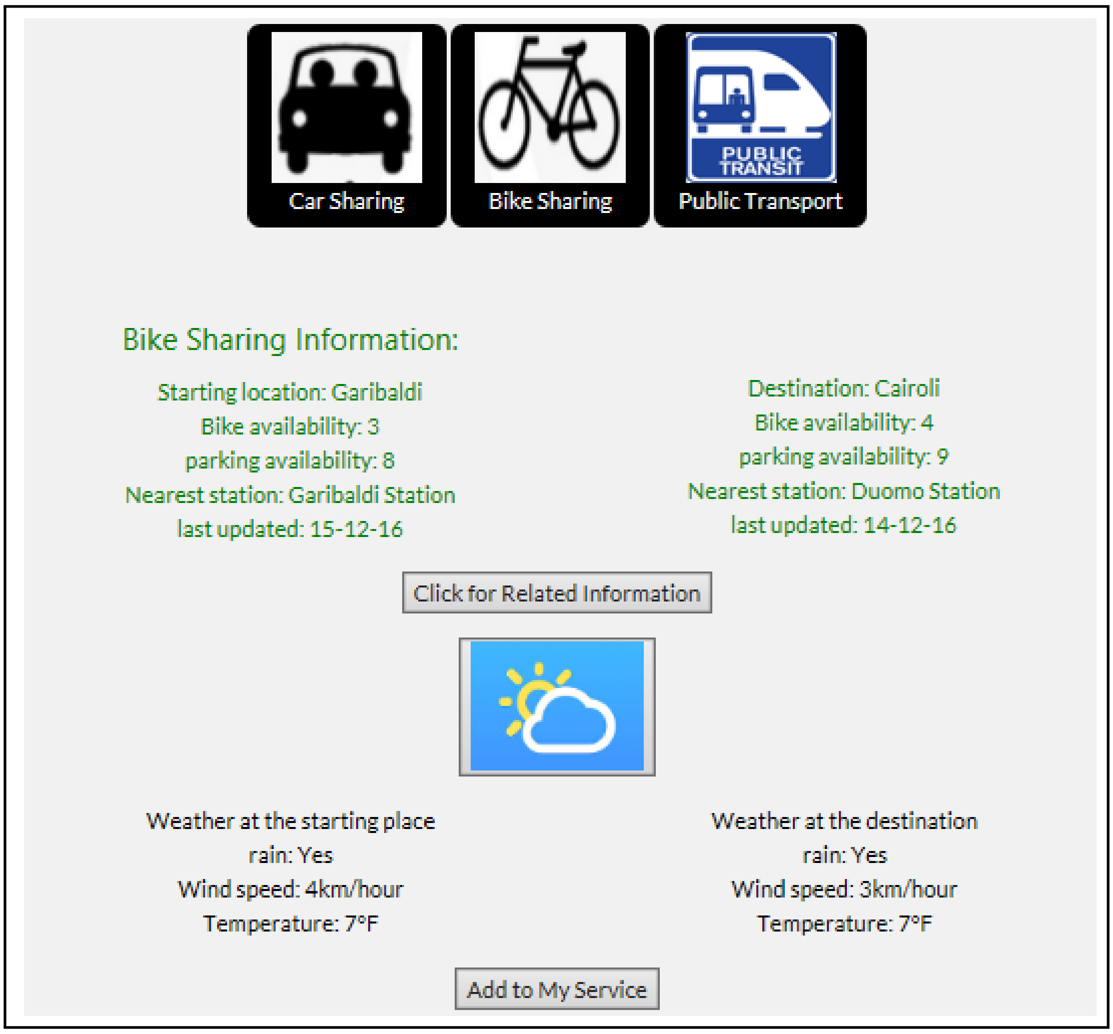
Destination: Cairoli
Bike availability: 4
parking availability: 9
Nearest station: Duomo Station
last updated: 14-12-16

Figure 8.11: Composition through a link

invoking a service that provides dynamic information about the elevator. To complete this task, the composition between the public transport API and the elevator API has been done (Fig. 8.14) at the back-end through the use of enriched descriptions.

8.4 Evaluation: Usability Testing of the Prototype Using System Usability Scale

In this section, we present the usability of our implemented prototype. We adopted the System Usability Scale (SUS) testing method to determine the eligibility of this application. This type of test is conducted to evaluate the usability of the systems. As one of our goals is to develop an application that can be easily adapted by all groups of end-users, we conducted this evaluation test to measure how extent our application is usable to the end-users.



Bike Sharing Information:

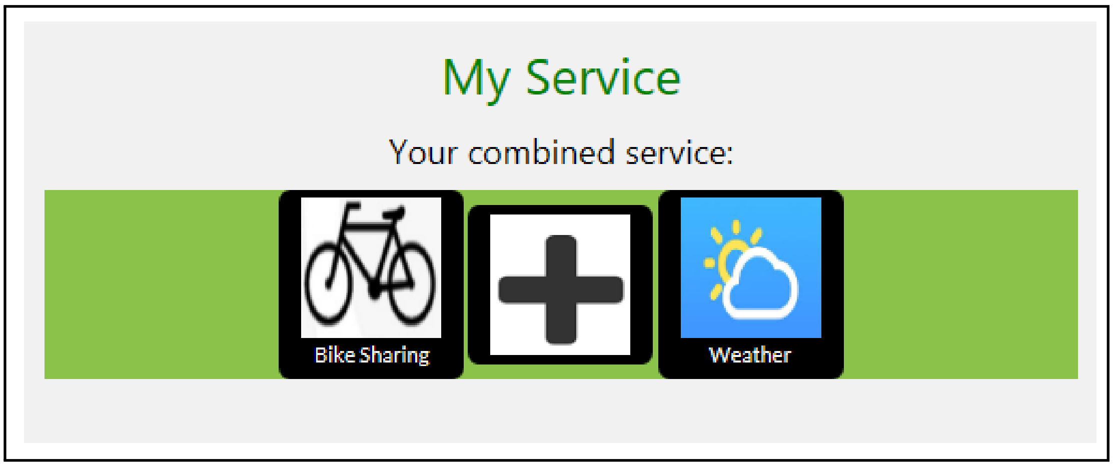
Starting location: Garibaldi
Bike availability: 3
parking availability: 8
Nearest station: Garibaldi Station
last updated: 15-12-16

Destination: Cairoli
Bike availability: 4
parking availability: 9
Nearest station: Duomo Station
last updated: 14-12-16

Weather at the starting place
rain: Yes
Wind speed: 4km/hour
Temperature: 7°F

Weather at the destination
rain: Yes
Wind speed: 3km/hour
Temperature: 7°F

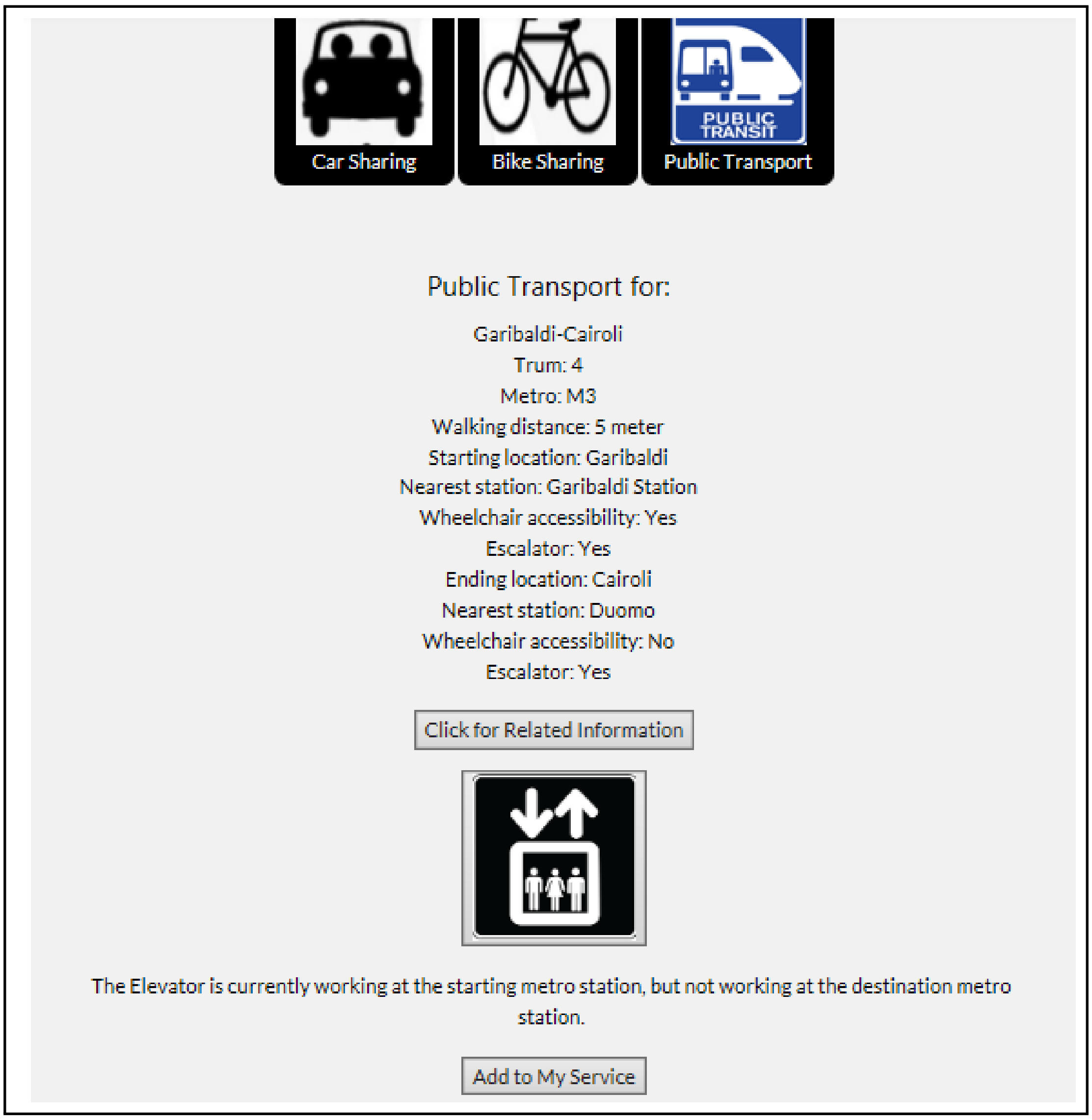
Figure 8.12: Bike Sharing service and Weather service composition



My Service

Your combined service:

Figure 8.13: My Service



Public Transport for:

Garibaldi-Cairoli

Trum: 4

Metro: M3

Walking distance: 5 meter

Starting location: Garibaldi

Nearest station: Garibaldi Station

Wheelchair accessibility: Yes

Escalator: Yes

Ending location: Cairoli

Nearest station: Duomo

Wheelchair accessibility: No

Escalator: Yes

The Elevator is currently working at the starting metro station, but not working at the destination metro station.

Figure 8.14: Public transportation and Elevator service composition

8.4.1 Methodology

The SUS scores are between 0 and 100, and they are analysed using percentiles. Percentiles are useful for giving the relative standing of an individual in a group, and they provide normalized ranks. A SUS score of 68 (a usability threshold) or higher signifies that a system is usable to its users [11]. The SUS is a simple, ten-item scale giving a global view of subjective assessments of the system's usability.

We conducted two evaluation tests that involved 24 end-users. We divided them in 2 groups for these evaluations. Before the evaluation process we interacted with the respondents to let them understand how our prototypical application works. Also we gathered some information about their familiarity with Information Technology or applications they use. In the first evaluation test, there were 12 respondents where 8 respondents were in the age between 22-54 and 4 respondents were in the age between 55-75. All of them were not having any knowledge about Web service or Web API composition and they are used to spend time on the internet for using social media services such as Facebook, Twitter etc. or using some applications like Skype, Imo, Weather services, Transport services such as ATM, Moovit etc.

To understand the effectiveness of our prototype among elderly end-users we executed another evaluation test with other 12 users who are in the age between 55-75. All of them were not having any knowledge about Web service or Web API composition. Although they are active on social media services such as Facebook and frequently using applications like Skype, Imo etc. for communicating with friends and family, they are not frequent in using Web applications for mobility or other purposes. Among these end-users only 4 respondents were familiar with Web applications related to weather, mobility and entertainment, while others were having very little knowledge about Web applications.

During the evaluation process all of the respondents were allowed to interact with our implemented prototype application, thereafter before any discussion took place, the respondents were asked to record their immediate response to each item (a SUS questionnaire found in Appendix A). The results of the 1st evaluation test are showed in Table 8.1 and Table 8.2. Moreover, the results of the 2nd evaluation test are showed in Table 8.3 and Table 8.4.

8.4.2 Results of the 1st SUS Evaluation Test

From this evaluation, we obtained a result that shows the overall SUS score 76.875 (Table 8.1). As this score is higher than the threshold SUS Score that is 68, it signifies that our prototype application is usable to all groups of end-users. To explore more information about the users' responses, we analyse the results of the evaluation test by normalizing all of the responses of the 12 respondents. Table 8.2 shows the responses from the users, for each question in the SUS. Since the SUS evaluation method has ten questions, each question is denoted by qn for n= 1...10 (i.e., it can be found in appendix A). The purpose of this analysis is to understand users' attitude and behaviour towards our prototype.

Table 8.1: SUS score

Users	**User's Score**	**SUS Score=(User's Score * 2.5)**
1	29	72.5
2	28	70
3	34	85
4	32	80
5	34	85
6	25	62.5
7	28	70
8	34	85
9	31	77.5
10	28	70
11	31	77.5
12	35	87.5
Sum	369	922.5
Overall SUS score		**76.875**

Table 8.2: Users' responses

SUS Question no.	**Agreed**	**Undecided**	**Disagreed**
q1	6	3	3
q2	1	1	10
q3	10	2	0
q4	1	1	10
q5	9	2	1
q6	0	1	11
q7	11	1	0
q8	1	0	11
q9	7	3	2
q10	1	0	11

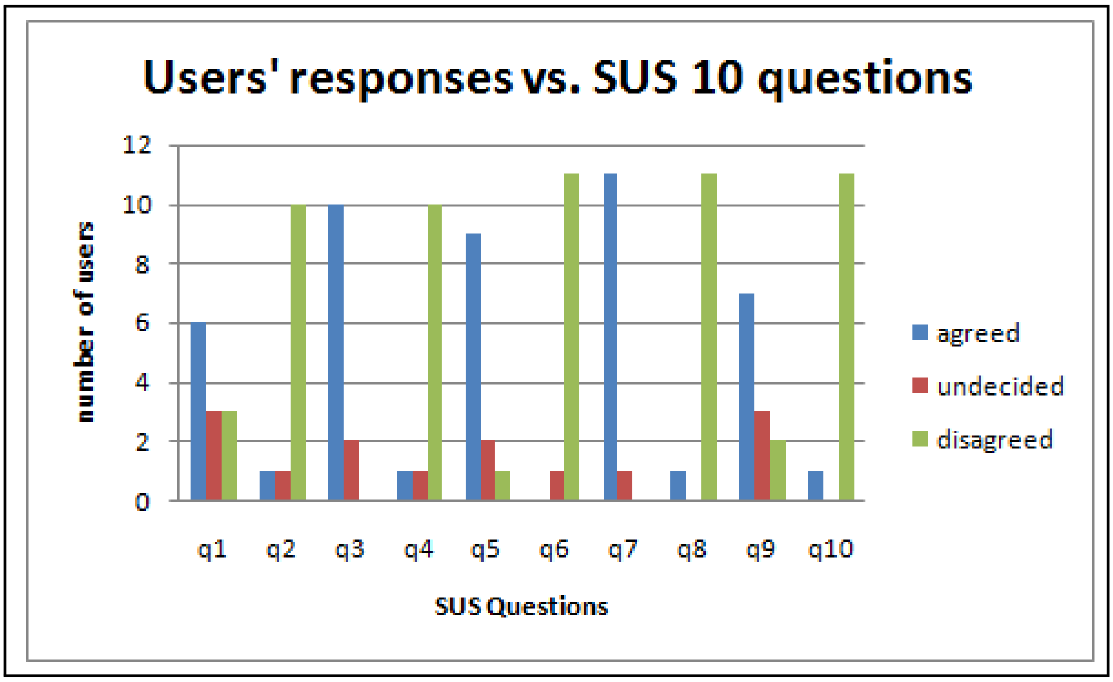


Figure 8.15: Users' responses on SUS questions

Table 8.2 and Fig. 8.15, shows the number of users who disagreed, undecided and agreed, for each question. From the users' responses, we consider both the *strongly agreed* and *agreed* as agreed, both *strongly disagreed* and *disagreed* as disagreed and *in-between* responses are considered as undecided. From Fig. 8.15, it can be seen that there were users who were in-between (i.e., undecided) for all the questions except question 8 and 10, while in other ones they agreed and disagreed. For all of the SUS questions, the users' responses were positive as explained in the following:

1. For q1, 6 respondents agreed to use this application frequently.
2. For q2, 10 respondents disagreed that they find this application unnecessarily complex, signifies that this application was not complex for them.
3. For q3, 10 respondents found this application as easy to use.
4. For q4, 10 respondents disagreed that they need the support of a technical person to be able to use this application.
5. For q5, 9 respondents agreed that the various functions in this application were well integrated.
6. For q6, 11 respondents disagreed that there was too much inconsistency in this application.

7. For q7, 11 respondents agreed that most people would learn to use this application very quickly.

8. For q8, 11 respondents disagreed that the application was very cumbersome to use.

9. For q9, 7 respondents felt very confident using this application.

10. For q10, 11 respondents disagreed that they had to learn a lot of things before they could use this application.

8.4.3 Results of the 2nd SUS Evaluation Test (Elderly Participants)

As one of our goals is to accommodate the elder people in the end-user development approach, we conducted this evaluation test considering only elderly end-users who are in the age between 55-75. From this evaluation, we obtained a result that shows the overall SUS score 72.083 (Table 8.3). As this score is higher than the threshold SUS Score that is 68, it signifies that our prototype application is usable to this specific group of end-users. We also analyse the results of the evaluation test by normalizing all of the responses of these 12 elderly respondents. In this way, we explore more information about these users considering their perceptions towards our prototype.

Fig. 8.16 and Table 8.4, shows the number of users who disagreed, undecided and agreed, for each question. From the users' responses, we consider both the *strongly agreed* and *agreed* as agreed, both *strongly disagreed* and *disagreed* as disagreed and *in-between* responses are considered as undecided. While for all of the SUS questions the end-users' responses were positive, for q4 and q7 more respondents were in-between (i.e., undecided). Further details as showed in Fig. 8.16, are explained in the following:

1. q1 shows that 9 respondents agreed to use this application frequently, that signifies they were satisfied with how the application operated.

2. q2 shows that, this application was not complex for 11 respondents.

3. q3 shows that 10 respondents found this application as easy to use.

4. For q4, 5 respondents disagreed that they would need the support of a technical person to be able to use this application. While only 1 respondent agreed that he would need a technical person to help him, 6 of the participants were in between (i.e. undecided).

5. For q5, 10 respondents agreed that the various functions in this application were well integrated.

6. For q6, 11 respondents disagreed that there was too much inconsistency in this application.

Table 8.3: SUS score for elderly participants

Users	**User's Score**	**SUS Score=(User's Score * 2.5)**
1	31	77.5
2	29	72.5
3	24	60
4	30	75
5	32	80
6	29	72.5
7	32	80
8	33	82.5
9	23	57.5
10	30	75
11	28	70
12	25	62.5
Sum	346	865
Overall SUS score		**72.083**

Table 8.4: Users' responses (elderly participants)

SUS Question no.	**Agreed**	**Undecided**	**Disagreed**
q1	9	2	1
q2	0	1	11
q3	10	1	1
q4	1	6	5
q5	10	1	1
q6	0	1	11
q7	5	7	0
q8	2	1	9
q9	5	5	2
q10	1	2	9

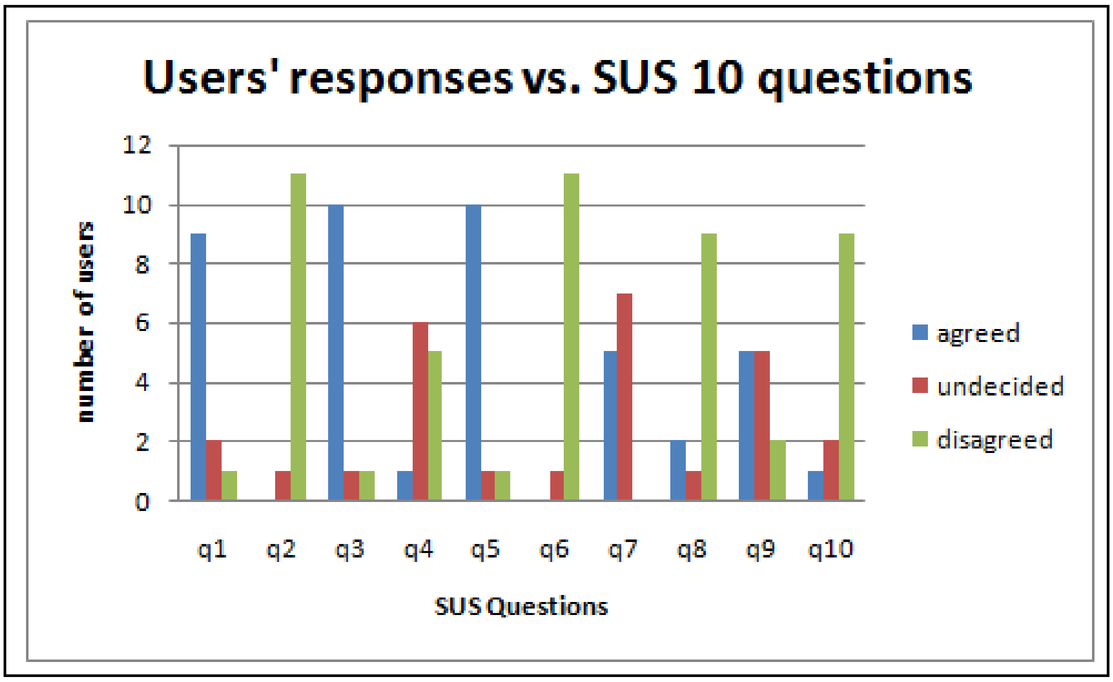


Figure 8.16: Users' responses on SUS questions with elderly participants

7. For q7, while 5 respondents could imagine that most people would learn to use this application very quickly, 7 of the respondents were in-between (i.e. undecided) about the learnability of others.

8. q8 shows that for 9 respondents this application was not very cumbersome to use.

9. For q9, 5 respondents felt very confident using this application. While 2 of the respondents were not confident, other 5 respondents were in-between (i.e. undecided) about their confidence in using this application.

10. For q10 which is the question on learnability of the application, the respondents showed that there was not much to learn recording a total of 9 out of 12.

8.5 Summary

This chapter describes the approaches we followed for bridging the gap between the technological constraints and all groups of end-users to adapt them in the Web API composition environment. We describe the technology stacks that are provided to the *Composition Editor* to facilitate the end-users to act as a developer without knowing any technological details. We also describe the usability features that we considered for accommodating all groups of end-users in this composition environment through the use of metaphors and usability guidelines.

After describing the prototype's features, we evaluate it using SUS evaluation method. We involved 24 end-users in two different evaluation tests. The 1st evaluation was done with 12 end-users from different age groups who are not expert in application developments or not having any programming knowledge. The result shows that our prototype is usable to all groups of end-users. The 2nd evaluation test was done with 12 elderly participants who have little knowledge about Web applications. Although the total SUS score of the 2nd evaluation test (i.e. 72.083) is less than the 1st evaluation test (i.e.76.875), our prototypical solution is also accepted by this specific group of end-users. Our detailed analysis on each of the SUS questions implies that the reason behind this reduction is the lack of confidence of elderly participants in using new applications or their dependency on others which are the common attitudes of older people as addressed in chapter 5. In the next chapter, we will conclude our research works.

Chapter 9

Conclusion

In this chapter, we discuss the research summary of this dissertation. We address the research questions and the corresponding results we achieved from our research works. We also outline the limitations and challenges we faced during our research and discuss our targeted future works.

We completed our research in a step by step process to set out a bridge that reduces the gap between the technological constraints and the end-users' requirements to achieve a user driven Web API composition platform. Considering the fact that, the value of personal relevance is a major driver for the successful adoption of new technologies, our analysis included the requirements of elder end-users regarding the use of ICT.

9.1 Research Summary

In this dissertation, we present a novel approach for user-driven composition of Web APIs by adopting end-user development paradigm. We dealt with three main research issues: i) heterogeneity of the Web services due to different protocols they use, (ii) automated addition of semantic annotations in the service descriptions and (iii) involvement of all groups of end-users in the composition process by providing an easy and accessible environment according to their requirements. By developing the solutions for resolving these issues we provided a platform for user-driven composition. We also analysed the requirements of a specific group of end-users (i.e. elders) to understand the context of this group of end-users. As end-users are more interested in using the services without knowing any technical details, we developed the prototype that represents the composite services without any technical terms and the users can develop their desired application very easily. Finally, our evaluation test showed that this prototypical tool is usable by all groups of end-users. The next section discusses the addressed research questions and the results of the analyses.

9.2 Addressed Research Questions and the Outcomes

In this section, we discuss the outcomes of our analyses by outlining our addressed research questions (chapter 1 section 1.2) for achieving the main goal of this dissertation "To facilitate the (semi) automatic Web API Composition through a platform by considering the end-users' requirements". To achieve this goal we divided our analyses to two different perspectives. One is to find the technological supports required to fulfill the (semi) automatic composition of Web services or Web APIs and another one is acquiring the requirements of end-users to accommodate them in the composition process. The main research question was "is it possible to provide an easy way to compose available Web APIs automatically according to user preferences?"

Considering this particular research question, our research was completed using the following main research hypotheses: Semantically enriched REST Web API descriptions can facilitate the (semi) automatic composition and an easy to use tool can facilitate the end-users to complete their required tasks. With the stated main question, we analysed three research sub questions that are discussed below:

How the Web Services and Web APIs interact with each other? To answer this question, we analysed different approaches for developing Web services such as SOAP, REST, OGC complaint SOS, SPS etc. We also analysed different composition approaches to understand how the Web services interact to automate the composition process. We found that REST is more appropriate for developing Web services, as it is a lightweight approach and the REST architectural style follows some constraints that enables the Web services to guide the users through links to go to the next state. Although there are existing approaches in the literature for automatic composition, a common way for communication among involved Web services is missing. Thus, we designed a framework for RESTful interactions between involved Web services that are using different protocols for communications [81]. We used HTTP protocol and design REST interfaces to make the Web services interoperable. Considering a scenario where sensors are acting as services, we designed RESTful interfaces for sensor services by defining URI schema and designed the hypermedia controlled interactions by using HTTP Idioms and Domain Application Protocol (DAP) for sensors. We also implemented our designed framework where we developed a REST interface that we termed as "REST wrapper" to interface a Cinema Portal that can facilitate the restful interactions between involved Web services. We encountered some implementation issues as we outlined in chapter 6 (section 6.5), but these implementations can be overcome by using new functionalities offered by new software frameworks.

Today's service technologies are marked by the proliferation of Web APIs, also called RESTful services when they conform to REST principles [82, 100] and general purpose Web APIs are provided by third parties to access underlying resources, functionalities or data, through web-based user interfaces. Moreover, our experiment on employing REST approach has shown its potential in the composition process. Thus, we consider Web APIs that are following REST approach in the rest of our research works.

How to provide a comprehensive set of information to facilitate (semi) automatic composition? To find a solution for this question, we analysed existing approaches for describing REST Web APIs. As there is no standard format for describing REST APIs, there exist several REST metadata formats. But these heterogeneous REST metadata formats are providing only functional information about the APIs (e.g. HTTP methods, URIs, model schema, etc.) but the information that qualifies the properties of APIs (e.g.classification of input arguments and response data) are missing. Moreover, the existing approaches for adding semantic annotation requires a lot of manual works. Thus, we proposed an approach for enriching Web API descriptions by adding semantic annotation (semi) automatically using Table Miner technique (discussed in chapter 7) [79]. We showed how semantic annotations can be added by adopting API profiles that links properties to concepts in shared vocabularies. To accomplish this work, we made an extension to the OAI specification. We developed editors targeting professional developers, where they can develop the Web APIs and its descriptions using the API Description Editor and can annotate the API descriptions and add these enriched descriptions to the Web APIs and can define the composition pattern to create Composite Web APIs by using API Annotation Editor. Then we showed that (semi) automatic composition can be done by using these enriched descriptions either by directly linking the outputs and inputs of selected APIs, or by including transformation services that transform and make outputs compatible to inputs according to the semantic relations hold in the annotations.

By providing the above discussed solutions, we resolved the addressed technological issues to facilitate Web API composition. To achieve our goal completely, we considered the end-users' requirements to accommodate them in the composition process. In particular, we analysed the following research question.

How the composition process can be easily accessible for all groups of end-users? To answer this question, we analysed different frameworks and tools that facilitate Web API composition or Mashups tools considering the end-users with little or no programming knowledge. We found that there are no approaches or tools that considered elders as end-users in the Web API composition environment. Although today's elders are more active on the internet and familiar with current information technologies, an easy way to search for information and utilizing these information to complete their own tasks is missing. Thus, we emphasis on understanding the needs and attitudes of a specific group of end-users e.g. elders as a case study, to answer the specific question *is it possible to exploit the opportunities of end-user development approach in a complex and an un-investigated context (e.g. elderly)?* We considered these elders as end-user developers in the End-user development paradigm. We also analysed different usability guidelines to design an easily accessible Web API composition tool that can be adapted by all groups of end-users including elders. From this analysis, we outlined a list of features for interface design of our prototypical solution (discussed in chapter 5). Then we developed the prototype considering these usability features and evaluated it using SUS testing method (discussed in chapter 8). The evaluation result showed that the prototype is acceptable to all groups of end-users.

9.3 Research Limitations and Challenges

In this section, we describe the limitations of our research works and some challenges that we encountered. To accomplish our work, we discovered different Web API descriptions manually. For example, we searched for weather APIs in ProgrammableWeb[1] and looked for its descriptions to the given links. For some Web APIs we had to even create or edit the descriptions manually. Moreover, we provided a limited set of Web APIs represented as services to the end-users. In the following we listed out some limitations and related challenges:

- Due to time constraints we addressed few vocabularies for correlation of properties, as correlation of different ontological concepts is itself a project about knowledge base.
- The composite services are created by the professional developers. Thus, the users are provided with a limited set of services that can not fulfill their different needs in different scenarios.
- Measuring usability also depends upon the users' or the respondents' personal experience on using similar applications, that may have effect during the evaluation process and on the obtained results. Furthermore the training given to users, might have been limited as compared to their mental capabilities.
- This research employed only user oriented evaluation methods as our target is to involve all groups of end-users, specially who are not having any programming knowledge or expert in developing applications. Moreover, apart from using SUS, also experiments to evaluate user experiences in a broader way might have more effects.
- In the evaluation, we have considered few elderly end-users. Testing our prototype with this specific group of end-users was a challenge as it was difficult to motivate them in this evaluation process.

9.4 Research Achievements

This research provides a platform for provisioning services and enable End-user development for all groups of end-users. To understand the requirements of end-users regarding our proposed

[1]https://www.programmableweb.it

composition platform, we did a requirement elicitation study analysing the opportunities of end-user development in a complex and an uninvestigated context (e.g. elder). This analysis resulted that, today's elders are active on the internet, using different Web sites or Web applications for different purposes. They are also interested in new technologies, devices or applications when these are presented in an easy to use manner.

Considering the interoperability issues, due to the existing plethora of heterogeneous Web services or Web APIs that are using different protocols or architectures, we designed and implemented RESTful interactions by employing RESTful interface and hypermedia controls that guide the users to facilitate automatic composition. Through the implementation of a REST wrapper to a legacy Web service we proved that RESTful interfaces that can facilitate interactions between involved Web services, thus facilitates composition.

Usually Web APIs are designed and developed to enable professional developers to use the developed services directly into their applications. The descriptions of these Web APIs are presented in textual formats that are not understandable by machines or even no descriptions at all. Although, there are some existing formats that are machine processable, a comprehensive set of information that can facilitate automatic composition is missing. Our proposed solution provides a way to enrich the descriptions with semantic annotations that can correlate properties at the semantic level and facilitates (semi) automatic composition.

With the advancement of Web technologies, there is a need to present the Web APIs in a way that can be accessible by all groups of users. We developed our user driven composition prototype by following a set of usability guidelines for implementing the interface, to make it accessible for all groups of end-users including elders. The usability evaluation test showed a positive results regarding the acceptability by all groups of end-users. Thus, we proved that it is possible to accommodate all groups of end-users in the End-user development paradigm, where these end-users are acting as *end-user developers* by using an easy to use tool.

9.5 Future Works

The outcomes of this study provide direction for more intense research on composing Web APIs automatically that considers elders as end-user developers by providing an abstract layer between the user interfaces and the technical details. The future research directs to enable the end-user developers to select their required services (i.e. Web APIs) from a pool of services and compose them automatically. To accomplish this, we may need to investigate new forms of interactions and define new metaphors to let end-users have a better understanding of the composition platform without having them to learn technical details.

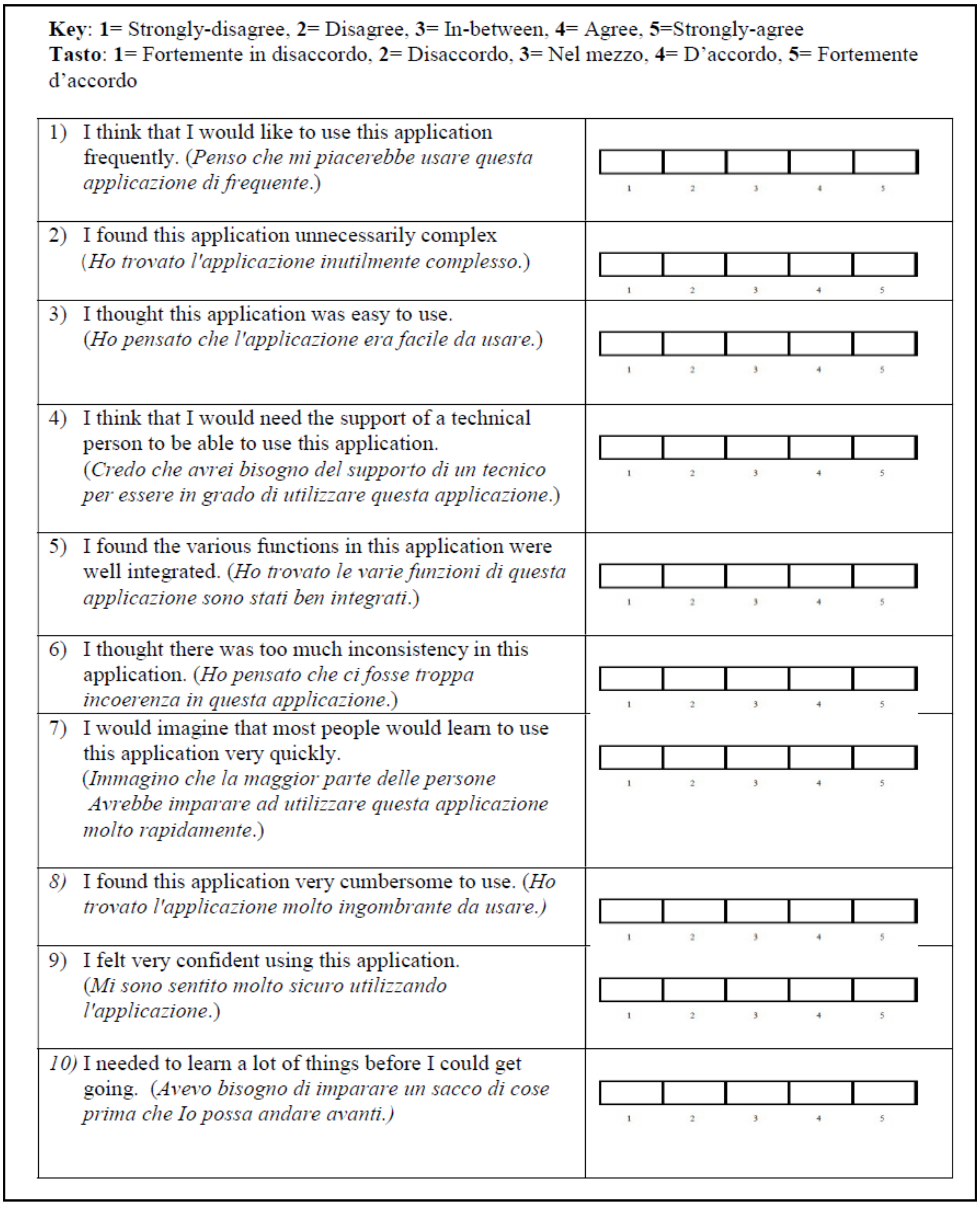
The limitation of this research directs to validate the prototype among more end-users including elders by adopting different experimental methods that would help in acquiring more requirements regarding end-user development environment and also help in improving the design in a much better way to support all groups of end-users. Moreover, we want to broaden our approach by using more concepts from shared vocabularies.

APPENDIX A

APPENDIX A

The SUS scale is generally used after the respondent has had an opportunity to use the system being evaluated, but before any debriefing or discussion takes place [11]. Respondents should be asked to record their immediate response to each item, rather than thinking about items for a long time. All items should be checked. If a respondent feels that they cannot respond to a particular item, they should mark the center point of the scale.

Note that scores for individual items are not meaningful on their own. To calculate the SUS score, first sum the score contributions from each item. Each item's score contribution will range from 0 to 4. For items 1, 3, 5, 7, and 9 the score contribution is the scale position minus 1. For items 2,4,6,8 and 10, the contribution is 5 minus the scale position. Multiply the sum of the scores by 2.5 to obtain the overall value of SUS. SUS scores have a range of 0 to 100.



Key: **1**= Strongly-disagree, **2**= Disagree, **3**= In-between, **4**= Agree, **5**=Strongly-agree
Tasto: **1**= Fortemente in disaccordo, **2**= Disaccordo, **3**= Nel mezzo, **4**= D'accordo, **5**= Fortemente d'accordo

1) I think that I would like to use this application frequently. (*Penso che mi piacerebbe usare questa applicazione di frequente.*)	☐ ☐ ☐ ☐ ☐ 1 2 3 4 5
2) I found this application unnecessarily complex (*Ho trovato l'applicazione inutilmente complesso.*)	☐ ☐ ☐ ☐ ☐ 1 2 3 4 5
3) I thought this application was easy to use. (*Ho pensato che l'applicazione era facile da usare.*)	☐ ☐ ☐ ☐ ☐ 1 2 3 4 5
4) I think that I would need the support of a technical person to be able to use this application. (*Credo che avrei bisogno del supporto di un tecnico per essere in grado di utilizzare questa applicazione.*)	☐ ☐ ☐ ☐ ☐ 1 2 3 4 5
5) I found the various functions in this application were well integrated. (*Ho trovato le varie funzioni di questa applicazione sono stati ben integrati.*)	☐ ☐ ☐ ☐ ☐ 1 2 3 4 5
6) I thought there was too much inconsistency in this application. (*Ho pensato che ci fosse troppa incoerenza in questa applicazione.*)	☐ ☐ ☐ ☐ ☐ 1 2 3 4 5
7) I would imagine that most people would learn to use this application very quickly. (*Immagino che la maggior parte delle persone Avrebbe imparare ad utilizzare questa applicazione molto rapidamente.*)	☐ ☐ ☐ ☐ ☐ 1 2 3 4 5
8) I found this application very cumbersome to use. (*Ho trovato l'applicazione molto ingombrante da usare.)*	☐ ☐ ☐ ☐ ☐ 1 2 3 4 5
9) I felt very confident using this application. (*Mi sono sentito molto sicuro utilizzando l'applicazione.*)	☐ ☐ ☐ ☐ ☐ 1 2 3 4 5
10) I needed to learn a lot of things before I could get going. (*Avevo bisogno di imparare un sacco di cose prima che Io possa andare avanti.)*	☐ ☐ ☐ ☐ ☐ 1 2 3 4 5

Figure A.1: SUS Questionnaire

BIBLIOGRAPHY

[1] S. M. AFFONSO DE LARA, W. M. WATANABE, E. P. B. DOS SANTOS, AND R. P. FORTES, *Improving wcag for elderly web accessibility*, in Proceedings of the 28th ACM international conference on design of communication, ACM, 2010, pp. 175–182.

[2] L. AGELIGHT, *Interface design guidelines for users of all ages*, AgeLight, Clyde Hill, pp 7-12, Available at: http://www.agelight.com/webdocs/designguide.pdf, Accessed: 18/10/2016, (2001).

[3] S. AGHAEE AND C. PAUTASSO, *End-user development of mashups with naturalmash*, Journal of Visual Languages & Computing, 25 (2014), pp. 414–432.

[4] S. AGHAEE, C. PAUTASSO, AND A. DE ANGELI, *Natural end-user development of web mashups*, in 2013 IEEE Symposium on Visual Languages and Human Centric Computing, IEEE, 2013, pp. 111–118.

[5] A. AULA AND M. KÄKI, *Less is more in web search interfaces for older adults*, First Monday, 10 (2005).

[6] T. BERNERS-LEE, *W3 future directions. plenary talk. w3*, 1994.

[7] T. BERNERS-LEE, *Linked data. design issues for the world wide web*, World Wide Web Consortium. http://www. w3. org/DesignIssues/LinkedData. html, (2006).

[8] M. A. BLYTHE, A. F. MONK, AND K. DOUGHTY, *Socially dependable design: The challenge of ageing populations for hci*, Interacting with Computers, 17 (2005), pp. 672–689.

[9] A. BOTTARO, A. GÉRODOLLE, AND P. LALANDA, *Pervasive service composition in the home network*, in 21st International Conference on Advanced Information Networking and Applications (AINA'07), IEEE, 2007, pp. 596–603.

[10] M. BOTTS, G. PERCIVALL, C. REED, AND J. DAVIDSON, *Ogc® sensor web enablement: Overview and high level architecture*, in International conference on GeoSensor Networks, Springer, 2006, pp. 175–190.

[11] J. BROOKE, *Sus: a retrospective*, Journal of usability studies, 8 (2013), pp. 29–40.

[12] A. BRÖRING, C. STASCH, AND J. ECHTERHOFF, *Ogc sensor observation service interface standard. open geospatial consortium (ogc)*, 2012.

[13] O. CAMPBELL, *Designing for the elderly: Ways older people use digital technology differently*. https:ttp://www.smashingmagazine.com/2015/02/designing-digital-technology-for-theelderly/.
Accessed: 2016-05-24.

[14] C. CAPPIELLO, F. DANIEL, M. MATERA, M. PICOZZI, AND M. WEISS, *Enabling end user development through mashups: requirements, abstractions and innovation toolkits*, in International Symposium on End User Development, Springer, 2011, pp. 9–24.

[15] K. CHEN, A. CHAN, AND Q. MA, *Cell phone feature preferences among older adults: A paired comparison study*, Gerontechnology: international journal on the fundamental aspects of technology to serve the ageing society, (2014).

[16] S. Y. CHEN, J.-P. FAN, AND R. D. MACREDIE, *Navigation in hypermedia learning systems: experts vs. novices*, Computers in Human Behavior, 22 (2006), pp. 251–266.

[17] A. CHEVALIER, A. DOMMES, D. MARTINS, AND C. VALÉRIAN, *Searching for information on the web: Role of aging and ergonomic quality of website*, in International Conference on Human-Computer Interaction, Springer, 2007, pp. 691–700.

[18] J. CHIN AND W.-T. FU, *Interactive effects of age and interface differences on search strategies and performance*, in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, 2010, pp. 403–412.

[19] D. CHISNELL AND J. REDISH, *Designing web sites for older adults: Expert review of usability for older adults at 50 web sites*, vol. 1, AARP, 2005.

[20] E. CHRISTENSEN, F. CURBERA, G. MEREDITH, S. WEERAWARANA, ET AL., *Web services description language (wsdl) 1.1*, 2001.

[21] L. CLARK, R. DESISTO, J. HOLINCHECK, A. WHITE, A. KYTE, A. SARNER, B. LHEUREUX, B. GASSMAN, C. KLAPPICH, AND E. KOLSKY, *Hype cycle for software as a service*, Gartner Research, ID G, 141122 (2012).

[22] M. CONCI, F. PIANESI, AND M. ZANCANARO, *Useful, social and enjoyable: Mobile phone adoption by older people*, in IFIP Conference on Human-Computer Interaction, Springer, 2009, pp. 63–76.

[23] P. J. DANIELSEN AND A. JEFFREY, *Validation and interactivity of web api documentation*, in Web Services (ICWS), 2013 IEEE 20th International Conference on, IEEE, 2013, pp. 523–530.

[24] T. H. DAVENPORT, *Thinking for a living: how to get better performances and results from knowledge workers*, Harvard Business Press, 2013.

[25] J. DINET, E. BRANGIER, G. MICHEL, R. VIVIAN, S. BATTISTI, AND R. DOLLER, *Older people as information seekers: Exploratory studies about their needs and strategies*, in International Conference on Universal Access in Human-Computer Interaction, Springer, 2007, pp. 877–886.

[26] W. DRYTKIEWICZ, I. RADUSCH, S. ARBANOWSKI, AND R. POPESCU-ZELETIN, *prest: a rest-based protocol for pervasive systems*, in Mobile Ad-hoc and Sensor Systems, 2004 IEEE International Conference on, IEEE, 2004, pp. 340–348.

[27] W. DZIDA, *Mary beth rosson and john m. carroll: Usability engineering: Scenario-based development of human-computer interaction*, i-com Zeitschrift für interaktive und kooperative Medien, 1 (2002), p. 46.

[28] C. EUROPEA, *The 2015 ageing report: Underlying assumptions and projection methodologies*, 2014.

[29] T. FIDGEON, *Usability for older web users*, WebCredible, February, (2006).

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

[31] R. T. FIELDING, *Rest apis must be hypertext-driven*, Untangled musings of Roy T. Fielding. [Online]. Available at: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven, Accessed:22/10/2016, (2008).

[32] G. FISCHER, E. GIACCARDI, Y. YE, A. G. SUTCLIFFE, AND N. MEHANDJIEV, *Meta-design: a manifesto for end-user development*, Communications of the ACM, 47 (2004), pp. 33–37.

[33] H. FLETCHER, *The principles of inclusive design (they include you)*, Architecture and the built environment, CABE, 1 (2006).

[34] M. FOWLER, *Richardson maturity model: steps toward the glory of rest*, Online at http://martinfowler. com/articles/richardsonMaturityModel. html, Accessed: 22/10/2016, (2010).

[35] J. GAO AND A. KORONIOS, *Mobile application development for senior citizens*, in PACIS, 2010, p. 65.

[36] C. GHAOUI, *Encyclopedia of human computer interaction*, IGI Global, 2005.

[37] G. GHIANI, F. PATERNÒ, L. D. SPANO, AND G. PINTORI, *An environment for end-user development of web mashups*, International Journal of Human-Computer Studies, 87 (2016), pp. 38–64.

[38] P. GREGOR AND A. DICKINSON, *Cognitive difficulties and access to information systems: an interaction design perspective*, Universal Access in the Information Society, 5 (2007), pp. 393–400.

[39] M. GUDGIN, M. HADLEY, N. MENDELSOHN, J.-J. MOREAU, H. F. NIELSEN, A. KARMARKAR, AND Y. LAFON, *Soap version 1.2*, W3C recommendation, 24 (2003).

[40] A. R. GUIDO, A. F. DO PRADO, W. L. DE SOUZA, AND E. G. DA SILVA, *Supporting the development of user-driven service composition applications*, in Information Technology: New Generations, Springer, 2016, pp. 519–530.

[41] D. GUINARD, V. TRIFA, AND E. WILDE, *A resource oriented architecture for the web of things*, in Internet of Things (IOT), 2010, IEEE, 2010, pp. 1–8.

[42] S. GUPTA, P. SZEKELY, C. A. KNOBLOCK, A. GOEL, M. TAHERIYAN, AND M. MUSLEA, *Karma: A system for mapping structured sources into the semantic web*, in Extended Semantic Web Conference, Springer, 2012, pp. 430–434.

[43] A. GUSTAVO, F. CASATI, H. KUNO, AND V. MACHIRAJU, *Web services: concepts, architectures and applications*, tech. rep., ISBN 3-540-44008-9, 2004.

[44] M. HADLEY, *Web application description language, w3c member submission 31 august 2009*, URL: http://www. w3. org/Submission/wadl/, Accessed: 28/09/ 2016, (2009).

[45] S. HANDSCHUH, S. STAAB, AND R. STUDER, *Leveraging metadata creation for the semantic web with cream*, in Ki 2003: Advances in Artificial Intelligence, Springer, 2003, pp. 19–33.

[46] B. HARTMANN, L. WU, K. COLLINS, AND S. R. KLEMMER, *Programming by a sample: rapidly creating web applications with d. mix*, in Proceedings of the 20th annual ACM symposium on User interface software and technology, ACM, 2007, pp. 241–250.

[47] S. HAUKKA, *Older australians and the internet*, ARC Centre of Excellence for Creative Industries and Innovation, (2011).

[48] D. HAWTHORN, *Training wheels for older users*, in Proceedings of the 17th Australia conference on computer-human Interaction: Citizens Online: Considerations for today and the future, Computer-Human Interaction Special Interest Group (CHISIG) of Australia, 2005, pp. 1–10.

[49] R. Heery and M. Patel, *Application profiles: mixing and matching metadata schemas*, Ariadne, (2000).

[50] C. Held, J. Kimmerle, and U. Cress, *Learning by foraging: The impact of individual knowledge and social tags on web navigation processes*, Computers in Human Behavior, 28 (2012), pp. 34–40.

[51] E. Hernández-Encuentra, M. Pousada, and B. Gómez-Zúñiga, *Ict and older people: Beyond usability*, Educational Gerontology, 35 (2009), pp. 226–245.

[52] R. J. Hodes and D. A. Lindberg, *Making your website senior friendly*, National Institute on Aging and the National Library of Medicine, (2002).

[53] B. Holt, *Creating senior-friendly web sites.*, Issue brief (Center for Medicare Education), 1 (2000), p. 1.

[54] J. L. Horn and R. B. Cattell, *Age differences in fluid and crystallized intelligence*, Acta psychologica, 26 (1967), pp. 107–129.

[55] D. F. Huynh, R. C. Miller, and D. R. Karger, *Enabling web browsers to augment web sites' filtering and sorting functionalities*, in Proceedings of the 19th annual ACM symposium on User interface software and technology, ACM, 2006, pp. 125–134.

[56] M. Jackson, *Software requirements & specifications: a lexicon of practice, principles and prejudices*, ACM Press/Addison-Wesley Publishing Co., 1995.

[57] I. Jacobs and N. Walsh, *Architecture of the world wide web*, W3C recommendation, 1 (2004).

[58] K. Janowicz, A. Bröring, C. Stasch, S. Schade, T. Everding, and A. Llaves, *A restful proxy and data model for linked sensor data*, International Journal of Digital Earth, 6 (2013), pp. 233–254.

[59] S. Jirka, A. Bröring, and C. Stasch, *Discovery mechanisms for the sensor web*, Sensors, 9 (2009), pp. 2661–2681.

[60] B. Johansson and M. Lahtinen, *Getting the balance right between functional and non-functional requirements: the case of requirement specification in it procurement*, International Journal of Information Systems and Project Management, 1 (2013), pp. 5–16.

[61] M. Johansson and M. Arvola, *A case study of how user interface sketches, scenarios and computer prototypes structure stakeholder meetings*, in Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI... but not as we know it-Volume 1, British Computer Society, 2007, pp. 177–184.

[62] L. KANTNER AND S. ROSENBAUM, *Usable computers for the elderly: applying coaching experiences*, in Professional Communication Conference, 2003. IPCC 2003. Proceedings. IEEE International, IEEE, 2003, pp. 10–pp.

[63] R. KHADKA AND B. SAPKOTA, *An evaluation of dynamic web service composition approaches*, in 4th International Workshop on Architectures Concepts and Technologies for Service-Oriented Computing (ACT4S0C'10), SciTePress, 2010, pp. 67–79.

[64] B. KLIMOVA, *Elderly people and their use of smart technologies: Benefits and limitations*, in Smart Education and e-Learning 2016, Springer, 2016, pp. 405–412.

[65] A. J. KO, R. ABRAHAM, L. BECKWITH, A. BLACKWELL, M. BURNETT, M. ERWIG, C. SCAFFIDI, J. LAWRANCE, H. LIEBERMAN, B. MYERS, ET AL., *The state of the art in end-user software engineering*, ACM Computing Surveys (CSUR), 43 (2011), p. 21.

[66] P. A. KOGUT AND W. S. HOLMES III, *Aerodaml: Applying information extraction to generate daml annotations from web pages.*, in Semannot@ K-CAP 2001, 2001.

[67] S. KURNIAWAN AND P. ZAPHIRIS, *Research-derived web design guidelines for older people*, in Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility, ACM, 2005, pp. 129–135.

[68] J. LAFFERTY, A. MCCALLUM, AND F. C. PEREIRA, *Conditional random fields: Probabilistic models for segmenting and labeling sequence data*, in Proceedings of the 18th International Conference on Machine Learning 2001 (ICML 2001), 2001, pp. 282–289.

[69] M. LANTHALER AND C. GÜTL, *A semantic description language for restful data services to combat semaphobia*, in Digital Ecosystems and Technologies Conference (DEST), 2011 Proceedings of the 5th IEEE International Conference on Digital Ecosystems and Technologies, IEEE, 2011, pp. 47–53.

[70] M. LANTHALER AND C. GUTL, *Hydra: A vocabulary for hypermedia-driven web apis.*, LDOW, 996 (2013).

[71] O. LASSILA AND R. R. SWICK, *Resource description framework (rdf) model and syntax specification*, W3C recommendation, (1999).

[72] A. L. LEMOS, F. DANIEL, AND B. BENATALLAH, *Web service composition: a survey of techniques and tools*, ACM Computing Surveys (CSUR), 48 (2016), p. 33.

[73] H. LIEBERMAN, F. PATERNÒ, M. KLANN, AND V. WULF, *End-user development: An emerging paradigm*, in End user development, Springer, 2006, pp. 1–8.

[74] X. LIU, Y. HUI, W. SUN, AND H. LIANG, *Towards service composition based on mashup*, in Services, 2007 IEEE Congress on, IEEE, 2007, pp. 332–339.

[75] D. LIZCANO, F. ALONSO, J. SORIANO, AND G. LÓPEZ, *A new end-user composition model to empower knowledge workers to develop rich internet applications*, Journal of Web Engineering, 10 (2011), pp. 197–233.

[76] ——, *A component-and connector-based approach for end-user composite web applications development*, Journal of Systems and Software, 94 (2014), pp. 108–128.

[77] ——, *Web-centred end-user component modelling*, Future Generation Computer Systems, 54 (2016), pp. 16–40.

[78] D. LIZCANO, J. SORIANO, M. REYES, AND J. J. HIERRO, *Ezweb/fast: reporting on a successful mashup-based solution for developing and deploying composite applications in the upcoming web of services*, in Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, ACM, 2008, pp. 15–24.

[79] M. N. LUCKY, M. CREMASCHI, B. LODIGIANI, A. MENOLASCINA, AND F. DE PAOLI, *Enriching api descriptions by adding api profiles through semantic annotation*, in International Conference on Service-Oriented Computing, Springer, 2016, pp. 780–794.

[80] M. N. LUCKY AND F. DEPAOLI, *Towards social inclusion of elder people using smart systems*, in Proceedings of The Challenge of Ageing Society: Technological Roles and Opportunities for Artificial Intelligence. Available at http://ceur-ws.org/Vol-1122, CEUR-WS, 2013.

[81] M. N. LUCKY, C. TZIVISKOU, AND F. DE PAOLI, *Towards restful communications in self-managing pervasive systems*, in International Conference on Service-Oriented Computing, Springer, 2012, pp. 263–274.

[82] P. A. LY, C. PEDRINACI, AND J. DOMINGUE, *Automated information extraction from web apis documentation*, in International Conference on Web Information Systems Engineering, Springer, 2012, pp. 497–511.

[83] M. MALESHKOVA, C. PEDRINACI, AND J. DOMINGUE, *Investigating web apis on the world wide web*, in Web Services (ECOWS), 2010 IEEE 8th European Conference on, IEEE, 2010, pp. 107–114.

[84] M. MATERA, M. PICOZZI, M. PINI, AND M. TONAZZO, *Peudom: a mashup platform for the end user development of common information spaces*, in International Conference on Web Engineering, Springer, 2013, pp. 494–497.

[85] L. MATHIASSEN, A. MUNK-MADSEN, P. A. NIELSEN, AND J. STAGE, *Objektorienterad analys och design*, Studentlitteratur, 1998.

[86] S. MAYER, N. INHELDER, R. VERBORGH, R. VAN DE WALLE, AND F. MATTERN, *Configuration of smart environments made simple: Combining visual modeling with semantic metadata and reasoning*, in Internet of Things (IOT), 2014 International Conference on the Internet of Things, IEEE, 2014, pp. 61–66.

[87] B. MEYER, R. A. SIT, V. A. SPAULDING, S. E. MEAD, AND N. WALKER, *Age group differences in world wide web navigation*, in CHI'97 Extended Abstracts on Human Factors in Computing Systems, ACM, 1997, pp. 295–296.

[88] R. MITRA, *Rapido: A sketching tool for web api designers*, in Proceedings of the 24th International Conference on World Wide Web Companion, International World Wide Web Conferences Steering Committee, 2015, pp. 1509–1514.

[89] B. A. MYERS, S. Y. JEONG, Y. XIE, J. BEATON, J. STYLOS, R. EHRET, J. KARSTENS, A. EFEOGLU, AND D. K. BUSSE, *Studying the documentation of an API for enterprise Service-Oriented Architecture*, IGI Global, 2012.

[90] H. NACER AND D. AISSANI, *Semantic web services: Standards, applications, challenges and solutions*, Journal of Network and Computer Applications, 44 (2014), pp. 134–151.

[91] J. NIELSEN AND K. P. COYNE, *Web usability for senior citizens*, Nielsen Norman Group, (2002).

[92] M. ODEH AND R. KAMM, *Bridging the gap between business models and system models*, Information and Software Technology, 45 (2003), pp. 1053–1060.

[93] R. PAK AND M. M. PRICE, *Designing an information search interface for younger and older adults*, Human Factors: The Journal of the Human Factors and Ergonomics Society, 50 (2008), pp. 614–628.

[94] F. PALMA, J. DUBOIS, N. MOHA, AND Y.-G. GUÉHÉNEUC, *Detection of rest patterns and antipatterns: a heuristics-based approach*, in International Conference on Service-Oriented Computing, Springer, 2014, pp. 230–244.

[95] L. PANZIERA, M. COMERIO, M. PALMONARI, F. DE PAOLI, AND C. BATINI, *Quality-driven extraction, fusion and matchmaking of semantic web api descriptions*, Journal of Web Engineering, 11 (2012), p. 247.

[96] L. PANZIERA AND F. DE PAOLI, *A framework for self-descriptive restful services*, in Proceedings of the 22nd international conference on World Wide Web companion, International World Wide Web Conferences Steering Committee, 2013, pp. 1407–1414.

[97] A. A. PATIL, S. A. OUNDHAKAR, A. P. SHETH, AND K. VERMA, *Meteor-s web service annotation framework*, in Proceedings of the 13th international conference on World Wide Web, ACM, 2004, pp. 553–562.

[98] C. PAUTASSO, *Restful web services: principles, patterns, emerging technologies*, in Web Services Foundations, Springer, 2014, pp. 31–51.

[99] C. PAUTASSO, O. ZIMMERMANN, AND F. LEYMANN, *Restful web services vs. big'web services: making the right architectural decision*, in Proceedings of the 17th international conference on World Wide Web, ACM, 2008, pp. 805–814.

[100] C. PEDRINACI, J. KOPECKỲ, M. MALESHKOVA, D. LIU, N. LI, AND J. DOMINGUE, *Unified lightweight semantic descriptions of web apis and web services*, W3C Workshop on Data and Services Integration, (2011).

[101] T. PHIRIYAPOKANON, *Is a big button interface enough for elderly users?: Towards user interface guidelines for elderly users*, Lambert Academic Publishing, 2011.

[102] I. PLAZA, L. MARTíN, S. MARTIN, AND C. MEDRANO, *Mobile applications in an aging society: Status and trends*, Journal of Systems and Software, 84 (2011), pp. 1977–1988.

[103] B. POPOV, A. KIRYAKOV, A. KIRILOV, D. MANOV, D. OGNYANOFF, AND M. GORANOV, *Kim–semantic annotation platform*, in The Semantic Web-ISWC 2003, Springer, 2003, pp. 834–849.

[104] A. POWELL, M. NILSSON, A. NAEVE, P. JOHNSTON, AND T. BAKER, *Dcmi abstract model. dcmi recommendation*, 2007.

[105] N. M. POWER, *A grounded theory of requirements documentation in the practice of software development*, PhD thesis, Dublin City University, 2002.

[106] J. RAO AND X. SU, *A survey of automated web service composition methods*, in International Workshop on Semantic Web Services and Web Process Composition, Springer, 2004, pp. 43–54.

[107] L. RICHARDSON AND S. RUBY, *RESTful web services*, " O'Reilly Media, Inc.", 2008.

[108] C. RODRÍGUEZ, M. BAEZ, F. DANIEL, F. CASATI, J. C. TRABUCCO, L. CANALI, AND G. PERCANNELLA, *Rest apis: A large-scale analysis of compliance with principles and best practices*, in International Conference on Web Engineering, Springer, 2016, pp. 21–39.

[109] D. ROMERO, G. HERMOSILLO, A. TAHERKORDI, R. NZEKWA, R. ROUVOY, AND F. ELIASSEN, *Restful integration of heterogeneous devices in pervasive environments*, in IFIP International Conference on Distributed Applications and Interoperable Systems, Springer, 2010, pp. 1–14.

[110] M. B. Rosson, J. Ballin, and J. Rode, *Who, what, and how: A survey of informal and professional web developers*, in 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05), IEEE, 2005, pp. 199–206.

[111] N. H. Rostami, E. Kheirkhah, and M. Jalali, *Web services composition methods and techniques: A review*, International Journal of Computer Science, Engineering & Information, (2013).

[112] M.-H. Ryu, S. Kim, and E. Lee, *Understanding the factors affecting online elderly user's participation in video ucc services*, Computers in Human Behavior, 25 (2009), pp. 619–632.

[113] S. Sayago and J. Blat, *A preliminary usability evaluation of strategies for seeking online information with elderly people*, in Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A), ACM, 2007, pp. 54–57.

[114] S. Sayago, D. Sloan, and J. Blat, *Everyday use of computer-mediated communication tools and its evolution over time: An ethnographical study with older people*, Interacting with Computers, 23 (2011), pp. 543–554.

[115] C. Scaffidi, M. Shaw, and B. Myers, *Estimating the numbers of end users and end user programmers*, in 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05), IEEE, 2005, pp. 207–214.

[116] M. Schmachtenberg, C. Bizer, and H. Paulheim, *Adoption of the linked data best practices in different topical domains*, in International Semantic Web Conference, Springer, 2014, pp. 245–260.

[117] C. Schroth and O. Christ, *Brave new web: Emerging design principles and technologies as enablers of a global soa*, in IEEE International Conference on Services Computing (SCC 2007), IEEE, 2007, pp. 597–604.

[118] R. Sease, *Metaphor's role in the information behavior of humans interacting with computers*, Information Technology and Libraries, 27 (2008), p. 9.

[119] N. Shadbolt, T. Berners-Lee, and W. Hall, *The semantic web revisited*, IEEE intelligent systems, 21 (2006), pp. 96–101.

[120] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, *Web services composition: A decade's overview*, Information Sciences, 280 (2014), pp. 218–238.

[121] A. P. Sheth, K. Gomadam, and J. Lathem, *Sa-rest: Semantically interoperable and easier-to-use services and mashups*, IEEE Internet Computing, 11 (2007), p. 91.

[122] I. SIMONIS AND J. ECHTERHOFF, *Ogc® sensor planning service implementation standard*, OpenGIS® implementation standard no. OGC, (2011), pp. 09–000.

[123] K. SLEGERS, M. VAN BOXTEL, AND J. JOLLES, *Effects of computer training and internet usage on cognitive abilities in older adults: a randomized controlled study*, Aging clinical and experimental research, 21 (2009), pp. 43–54.

[124] D. SLOAN, *Two cultures? the disconnect between the web standards movement and research-based web design guidelines for older people*, Gerontechnology, 5 (2006), pp. 106–112.

[125] T. STEINER AND J. ALGERMISSEN, *Fulfilling the hypermedia constraint via http options, the http vocabulary in rdf, and link headers*, in Proceedings of the second international workshop on RESTful design, ACM, 2011, pp. 11–14.

[126] A. STELLMAN AND J. GREENE, *Applied software project management*, " O'Reilly Media, Inc.", 2005.

[127] P. SUTER AND E. WITTERN, *Inferring web api descriptions from usage data*, in Hot Topics in Web Systems and Technologies (HotWeb), 2015 Third IEEE Workshop on, IEEE, 2015, pp. 7–12.

[128] M. TAHERIYAN, C. A. KNOBLOCK, P. SZEKELY, AND J. L. AMBITE, *Rapidly integrating services into the linked data cloud*, in The Semantic Web–ISWC 2012, Springer, 2012, pp. 559–574.

[129] M. TAHERIYAN, C. A. KNOBLOCK, P. SZEKELY, AND J. L. AMBITE, *Semi-automatically modeling web apis to create linked apis*, in Proceedings of the ESWC 2012 Workshop on Linked APIs, 2012.

[130] A. THATCHER, *Web search strategies: The influence of web experience and task type*, Information Processing & Management, 44 (2008), pp. 1308–1329.

[131] D. TOSI AND S. MORASCA, *Supporting the semi-automatic semantic annotation of web services: A systematic literature review*, Information and Software Technology, 61 (2015), pp. 16–32.

[132] R. TSOUROPLIS, M. PETYCHAKIS, I. ALVERTIS, E. BILIRI, F. LAMPATHAKI, AND D. ASKOUNIS, *Community-based api builder to manage apis and their connections with cloud-based services*, in CAiSE Forum, 2015.

[133] USCENSUS, *Us census bureau news: The older americans month facts for feature*. https://www.census.gov/newsroom/facts-for-features/2015/cb15-ff09.html. Accessed: 2016-05-24.

[134] P.-Y. VANDENBUSSCHE, G. A. ATEMEZING, M. POVEDA-VILLALÓN, AND B. VATANT, *Linked open vocabularies (lov): a gateway to reusable semantic vocabularies on the web*, Semantic Web, (2015), pp. 1–16.

[135] R. VERBORGH, D. ARNDT, S. VAN HOECKE, J. DE ROO, G. MELS, T. STEINER, AND J. GABARRO, *The pragmatic proof: Hypermedia api composition and execution*, Theory and Practice of Logic Programming, (2016), pp. 1–48.

[136] R. VERBORGH, A. HARTH, M. MALESHKOVA, S. STADTMÜLLER, T. STEINER, M. TAHERIYAN, AND R. VAN DE WALLE, *Survey of semantic description of rest apis*, in REST: Advanced Research Topics and Practical Applications, Springer, 2014, pp. 69–89.

[137] R. VERBORGH, E. MANNNENS, AND R. VAN DE WALLE, *Bottom-up web apis with self-descriptive responses*, in Proceedings of the First Karlsruhe Service Summit Workshop-Advances in Service Research, KIT Scientific Publishing, 2015, p. 143.

[138] R. VERBORGH, T. STEINER, D. VAN DEURSEN, J. DE ROO, R. VAN DE WALLE, AND J. G. VALLÉS, *Description and interaction of restful services for automatic discovery and execution*, in 2011 FTRA International workshop on Advanced Future Multimedia Services (AFMS 2011), FTRA, 2011.

[139] R. VERBORGH, T. STEINER, D. VAN DEURSEN, J. DE ROO, R. VAN DE WALLE, AND J. G. VALLÉS, *Capturing the functionality of web services with functional descriptions*, Multimedia tools and applications, 64 (2013), pp. 365–387.

[140] J. WALDO, G. WYANT, A. WOLLRATH, AND S. KENDALL, *A note on distributed computing*, in Mobile Object Systems Towards the Programmable Internet, Springer, 1997, pp. 49–64.

[141] J.-J. WANG AND A. S. KAUFMAN, *Changes in fluid and crystallized intelligence across the 20-to 90-year age range on the k-bit*, Journal of Psychoeducational Assessment, 11 (1993), pp. 29–37.

[142] J. WEBBER, S. PARASTATIDIS, AND I. ROBINSON, *REST in practice: Hypermedia and systems architecture*, " O'Reilly Media, Inc.", 2010.

[143] C. WICKENS, J. LEE, Y. LIU, AND S. BECKER, *Engineering anthropometry and workspace design*, An introduction to human factors engineering. 2nd ed. Upper Saddle River, NJ: Pearson Prentice Hall, (2004), pp. 243–268.

[144] E. WILDE, *The "profile" link relation type*.
https://www.ietf.org/rfc/rfc6906.txt.
Accessed: 2016-05-24.

[145] ——, *Linked data and service orientation*, in International Conference on Service-Oriented Computing, Springer, 2010, pp. 61–76.

[146] D. WRIGHT AND K. WADHWA, *Mainstreaming the e-excluded in europe: strategies, good practices and some ethical issues*, Ethics and Information Technology, 12 (2010), pp. 139–156.

[147] E. R. YEARBOOK, *Eurostat*, European Commission, (2012).

[148] M. ZAJICEK, *Special interface requirements for older adults*, Proceedings of the WUAUC, 1 (2001), pp. 22–25.

[149] ——, *Successful and available: interface design exemplars for older users*, Interacting with computers, 16 (2004), pp. 411–430.

[150] Z. ZHANG, *Start small, build complete: Effective and efficient semantic table interpretation using tableminer*, Under transparent review: The Semantic Web Journal, (2014).

[151] ——, *Towards efficient and effective semantic table interpretation*, in International Semantic Web Conference, Springer, 2014, pp. 487–502.

[152] H. ZHAO AND P. DOSHI, *Towards automated restful web service composition*, in Web Services, 2009. ICWS 2009. IEEE International Conference on, IEEE, 2009, pp. 189–196.