UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

SEMANTIC SERVICE DISCOVERY TECHNIQUES FOR THE COMPOSABLE WEB

TESIS DOCTORAL

JOSÉ IGNACIO FERNÁNDEZ VILLAMOR
Ingeniero de Telecomunicación

2012

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

SEMANTIC SERVICE DISCOVERY TECHNIQUES FOR THE COMPOSABLE WEB

TESIS DOCTORAL

JOSÉ IGNACIO FERNÁNDEZ VILLAMOR
Ingeniero de Telecomunicación

2012

Departamento de Ingeniería de Sistemas Telemáticos

Escuela Técnica Superior de Ingenieros de Telecomunicación

Universidad Politécnica de Madrid

Semantic Service Discovery Techniques for the Composable Web

Autor:

José Ignacio Fernández Villamor
Ingeniero de Telecomunicación

Tutores:

Carlos Ángel Iglesias Fernández
Doctor Ingeniero de Telecomunicación

Mercedes Garijo Ayestarán
Doctora Ingeniera de Telecomunicación

2012

Tribunal nombrado por el Magfco. y Excmo. Sr. Rector de la Universidad Politécnica de Madrid, el día 1 de junio de 2012.

Presidente: _____

Vocal: _____

Vocal: _____

Vocal: _____

Secretario: _____

Suplente: _____

Suplente: _____

Realizado el acto de defensa y lectura de la Tesis el día 1 de julio de 2012 en la E.T.S.I.T. habiendo obtenido la calificación de _____.

EL PRESIDENTE LOS VOCALES

EL SECRETARIO

A mis padres

Agradecimientos

Prefiero dar agradecimientos en persona, pero aceptaré inmortalizarlos en esta sección. Muy merecidamente le doy las gracias a mis tutores Carlos y Mercedes, que más allá de la gran dirección de tesis, han sabido crear un ambiente de trabajo extraordinario en el grupo. Desde el día en que entré, en el que era el único doctorando en el laboratorio, me he sentido como en casa y os lo agradezco enormemente. Gracias también al resto de profesores del grupo: Gregorio, José Carlos, Luis Enrique y Marifeli.

Por fortuna el grupo se repobló con gente magnífica. Gracias a Nacho, Álvaro, Dani, Paloma, Jorge, Laura, Elena, Adam, Paco, Miguel, Vicente, Álvaro, Jota y Geovanny por los días que hemos compartido juntos, tanto dentro como fuera del laboratorio, de los que me llevo algunos inolvidables. Gracias a ese estupendo elenco de futuras promesas que ha sido el plantel de becarios. Especialmente, por su contribución a la tesis, gracias a Alberto, Adriano, Pablo, Jacobo, Dani, Toni, Patricia, Adrián, Rubén y Danny. Y gracias a toda la gente del DIT con la que he trabajado; muy especialmente a Boni y Samuel.

Gracias a mis padres, mi hermano, mis abuelas, y también a mis abuelos, que ya no están, porque sin su compañía y sin la educación que me inculcaron no habría terminado un doctorado ni por casualidad.

Gracias a mis amigos, que son los más grandes: Rodrigo, Álvaro, Berta, Irene, Antonio, Esther, Luigi, Hesse, Javi, Mireia, Juan, Ana Alhama, Jorge, Ana y Dani, Pablo, Luis…

Und, natürlich, vielen Dank an die deutschen Leute aus Chemnitz. Es war wirklich sehr schön, mit euch zu arbeiten. Danke schön, Frank, Alexey, Hendrik und Martin. Und es war ganz toll, dass du drei Monate hier in Madrid gewohnt hast, Tilo. Ich hoffe, dass du bald wiederkommst. Alle sind willkommen.

Y sobre todo, mil disculpas a quien haya olvidado incluir aquí, que seguramente alguien habrá.

Abstract

This PhD thesis contributes to the problem of resource and service discovery in the context of the composable web. In the current web, mashup technologies allow developers reusing services and contents to build new web applications. However, developers face a problem of information flood when searching for appropriate services or resources for their combination.

To contribute to overcoming this problem, a framework is defined for the discovery of services and resources. In this framework, three levels are defined for performing discovery at content, discovery and agente levels.

The content level involves the information available in web resources. The web follows the Representational Stateless Transfer (REST) architectural style, in which resources are returned as representations from servers to clients. These representations usually employ the HyperText Markup Language (HTML), which, along with Content Style Sheets (CSS), describes the markup employed to render representations in a web browser. Although the use of Semantic Web standards such as Resource Description Framework (RDF) make this architecture suitable for automatic processes to use the information present in web resources, these standards are too often not employed, so automation must rely on processing HTML. This process, often referred as Screen Scraping in the literature, is the content discovery according to the proposed framework. At this level, discovery rules indicate how the different pieces of data in resources' representations are mapped onto semantic entities. By processing discovery rules on web resources, semantically described contents can be obtained out of them.

The service level involves the operations that can be performed on the web. The current web allows users to perform different tasks such as search, blogging, e-commerce, or social networking. To describe the possible services in RESTful architectures, a high-level feature-oriented service methodology is proposed at this level. This lightweight description framework allows defining service discovery

rules to identify operations in interactions with REST resources. The discovery is thus performed by applying discovery rules to contents discovered in REST interactions, in a novel process called service probing. Also, service discovery can be performed by modelling services as contents, i.e., by retrieving Application Programming Interface (API) documentation and API listings in service registries such as Programmable Web. For this, a unified model for composable components in Mashup-Driven Development (MDD) has been defined after the analysis of service repositories from the web.

The agent level involves the orchestration of the discovery of services and contents. At this level, agent rules allow to specify behaviours for crawling and executing services, which results in the fulfilment of a high-level goal. Agent rules are plans that allow introspecting the discovered data and services from the web and the knowledge present in service and content discovery rules to anticipate the contents and services to be found on specific resources from the web. By the definition of plans, an agent can be configured to target specific resources.

The discovery framework has been evaluated on different scenarios, each one covering different levels of the framework. Contenidos a la Carta project deals with the mashing-up of news from electronic newspapers, and the framework was used for the discovery and extraction of pieces of news from the web. Similarly, in Resulta and VulneraNET projects the discovery of ideas and security knowledge in the web is covered, respectively. The service level is covered in the OMELETTE project, where mashup components such as services and widgets are discovered from component repositories from the web. The agent level is applied to the crawling of services and news in these scenarios, highlighting how the semantic description of rules and extracted data can provide complex behaviours and orchestrations of tasks in the web.

The main contributions of the thesis are the unified framework for discovery, which allows configuring agents to perform automated tasks. Also, a scraping ontology has been defined for the construction of mappings for scraping web resources. A novel first-order logic rule induction algorithm is defined for the automated construction and maintenance of these mappings out of the visual information in web resources. Additionally, a common unified model for the discovery of services is defined, which allows sharing service descriptions.

Future work comprises the further extension of service probing, resource ranking, the extension of the Scraping Ontology, extensions of the agent model, and contructing a base of discovery rules.

Resumen

La presente tesis doctoral contribuye al problema de descubrimiento de servicios y recursos en el contexto de la web combinable. En la web actual, las tecnologías de combinación de aplicaciones permiten a los desarrolladores reutilizar servicios y contenidos para construir nuevas aplicaciones web. Pese a todo, los desarrolladores afrontan un problema de saturación de información a la hora de buscar servicios o recursos apropiados para su combinación.

Para contribuir a la solución de este problema, se propone un marco de trabajo para el descubrimiento de servicios y recursos. En este marco, se definen tres capas sobre las que se realiza descubrimiento a nivel de contenido, servicio y agente.

El nivel de contenido involucra a la información disponible en recursos web. La web sigue el estilo arquitectónico Representational Stateless Transfer (REST), en el que los recursos son devueltos como representaciones por parte de los servidores a los clientes. Estas representaciones normalmente emplean el lenguaje de marcado HyperText Markup Language (HTML), que, unido al estándar Content Style Sheets (CSS), describe el marcado empleado para mostrar representaciones en un navegador web. Aunque el uso de estándares de la web semántica como Resource Description Framework (RDF) hace apta esta arquitectura para su uso por procesos automatizados, estos estándares no son empleados en muchas ocasiones, por lo que cualquier automatización debe basarse en el procesado del marcado HTML. Este proceso, normalmente conocido como *Screen Scraping* en la literatura, es el descubrimiento de contenidos en el marco de trabajo propuesto. En este nivel, un conjunto de reglas de descubrimiento indican cómo los diferentes datos en las representaciones de recursos se corresponden con entidades semánticas. Al procesar estas reglas sobre recursos web, pueden obtenerse contenidos descritos semánticamente.

El nivel de servicio involucra las operaciones que pueden ser llevadas a cabo en la web. Actualmente, los usuarios de la web pueden realizar diversas tareas como

búsqueda, *blogging*, comercio electrónico o redes sociales. Para describir los posibles servicios en arquitecturas REST, se propone en este nivel una metodología de alto nivel para descubrimiento de servicios orientada a funcionalidades. Este marco de descubrimiento ligero permite definir reglas de descubrimiento de servicios para identificar operaciones en interacciones con recursos REST. Este descubrimiento es por tanto llevado a cabo al aplicar las reglas de descubrimiento sobre contenidos descubiertos en interacciones REST, en un nuevo procedimiento llamado sondeo de servicios. Además, el descubrimiento de servicios puede ser llevado a cabo mediante el modelado de servicios como contenidos. Es decir, mediante la recuperación de documentación de Application Programming Interfaces (APIs) y listas de APIs en registros de servicios como Programmable Web. Para ello, se ha definido un modelo unificado de componentes combinables para Mashup-Driven Development (MDD) tras el análisis de repositorios de servicios de la web.

El nivel de agente involucra la orquestación del descubrimiento de servicios y contenidos. En este nivel, las reglas de nivel de agente permiten especificar comportamientos para el rastreo y ejecución de servicios, lo que permite la consecución de metas de mayor nivel. Las reglas de los agentes son planes que permiten la introspección sobre los datos y servicios descubiertos, así como sobre el conocimiento presente en las reglas de descubrimiento de servicios y contenidos para anticipar contenidos y servicios por encontrar en recursos específicos de la web. Mediante la definición de planes, un agente puede ser configurado para descubrir recursos específicos.

El marco de descubrimiento ha sido evaluado sobre diferentes escenarios, cada uno cubriendo distintos niveles del marco. El proyecto Contenidos a la Carta trata de la combinación de noticias de periódicos digitales, y en él el framework se ha empleado para el descubrimiento y extracción de noticias de la web. De manera análoga, en los proyectos Resulta y VulneraNET se ha llevado a cabo un descubrimiento de ideas y de conocimientos de seguridad, respectivamente. El nivel de servicio se cubre en el proyecto OMELETTE, en el que componentes combinables como servicios y *widgets* se descubren en repositorios de componentes de la web. El nivel de agente se aplica al rastreo de servicios y noticias en estos escenarios, mostrando cómo la descripción semántica de reglas y datos extraídos permiten proporcionar comportamientos complejos y orquestaciones de tareas en la web.

Las principales contribuciones de la tesis son el marco de trabajo unificado para descubrimiento, que permite configurar agentes para realizar tareas automatizadas.

Además, una ontología de extracción ha sido definida para la construcción de correspondencias y extraer información de recursos web. Asimismo, un algoritmo para la inducción de reglas de lógica de primer orden se ha definido para la construcción y el mantenimiento de estas correspondencias a partir de la información visual de recursos web. Adicionalmente, se ha definido un modelo común y unificado para el descubrimiento de servicios que permite la compartición de descripciones de servicios.

Como trabajos futuros se considera la extensión del sondeo de servicios, clasificación de recursos, extensión de la ontología de extracción y la construcción de una base de reglas de descubrimiento.

Contents

Chapter 1

Introduction

In this chapter, the motivation and objectives of the thesis are introduced to the reader. The current web contains a vast amount of information in web applications. Additionally, web applications usually publish their services so that third-party applications can consume them. Through the use of standards like Resource Description Framework (RDF) proposed by the Semantic Web applications can consume the contents and use the services automatically. However, very frequently semantic descriptions of contents and services are not used, which limits the construction of these kinds of applications. This thesis contributes to the problem of bootstrapping semantic descriptions in the web by defining a uniform framework for discovery of contents and services.

1.1 Motivation

The web has an increasing number of applications and services that cover different domains and fields. Users can enjoy a wide range of applications, from e-commerce to blogging, media or social networking. The possibilities of the current web are only limited by the interoperability between applications. Internet versatility would increase if applications could be arbitrarily composed and automatically executed to fulfil a user's goal. This Service-Oriented Architecture (SOA) approach [Erl, 2005] has led to research in several areas, such as Semantic Web Services [Zhou et al., 2006] or mashups [Yu et al., 2008].

SOA suggests services being publicly accessible to make the Web a kind of global software library, with software services and components ready for their use by developers. In order to make services machine processable, Semantic Web Services [Zhou et al., 2006] standards such as OWL-S [World Wide Web Consortium, 2004] or Web Service Modeling Ontology (WSMO) [Roman et al., 2005] allow building semantic service descriptions.

The Web Services architecture misses integration with Representational Stateless Transfer (REST) architectural style [Wilde and Gaedke, 2008], resulting in the use of the Web as a platform and thus adding unnecessary complexity to the protocol stack. In order to improve the integration of the service architecture with the REST architectural style of the Web, initiatives such as Web Application Description Language (WADL) [Hadley, 2009] try to enable Semantic Web Services approach with a RESTful design in mind.

Anyway, both Web Services-based and REST-based Semantic Web Services approaches share the common target of defining semantically rich service descriptions in order to enable agents to perform automatic tasks such as service discovery, execution, and composition.

Such approaches for semantic description provide means to describe every kind of service and process the respective descriptions to automate tasks such as execution or composition. They are abstract and flexible enough to allow describing every possible service. Whenever a new service is deployed, a description has to be built in order to make the service available for automatic agents to process it. Usually, this implies describing service's inputs and outputs semantically, as well as defining the precise operation the service performs, which can be a time-consuming task.

As a result, although these kinds of approaches are formally sound and many

have been successful from a research point of view, none has reached wide adoption in the industry [Murphy et al., 2008]. The effort of describing services in web applications can be too big and too often web developers do not perform such task.

Similarly, the contents that are published in web resources experience a similar situation. Semantic technologies such as Resource Description Framework in Attributes (RDFa) [Adida and Birbeck, 2008] allow annotating web resources and publishing Linked Data [Bizer et al., 2009], which makes information machine processable. However, the lack of Semantic Web applications that exploit annotated contents is small, resulting in content providers not annotating their contents semantically.

Still, even with semantically annotated contents, there are pieces of information which are either hidden behind search services (or, in general, any HyperText Markup Language (HTML) form) or hardly accessible, so long as information discovery in the Web is performed through web spidering. This makes that tasks such as getting a particular archive of posts in a web log is a tough task that requires spidering techniques such as focused crawling [Chakrabarti et al., 1999] or using search forms. By describing search services semantically, access to information is improved.

1.2 Objectives

As mentioned, the Semantic Web experiences a bootstrapping problem because of the reduced amount of semantically described services and contents. Applications that exploit semantically annotated services and contents are scarce, while annotated resources do not grow because this scarcity of semantically empowered applications. This chicken and egg problem can be addressed through automatic resource discovery, which is the main topic of the thesis.

The thesis has the following objectives:

- Define a unified discovery framework for REST architectural style. The web follows the REST architectural style, which consists of a stateless hypermedia system comprisen by resources with uniform interfaces. The possible interactions with these resources are the services available in the system, while representations include the possible contents available in the web. Some approaches to discovery and semantic description dismiss these design guidelines of the web. This thesis attempts to define a framework

that fits the REST architectural style of the web while providing unified techniques to perform discovery at content and service levels.

- Research techniques for automatic content discovery. Plenty of the resources available in the web are not annotated. Therefore, their representations contain unstructured contents which require techniques for their extraction and discovery. The thesis attempts to contribute to information extraction by defining models and algorithms for content discovery.

- Research techniques for automatic service discovery. In the REST architectural style, interactions with resources follow a uniform interface, whereby a user agent is able to perform a predictable action on a resource. However, aspects such as HyperText Transfer Protocol (HTTP) parameters, the specific domain of the web resource which receives the interaction, or the loose semantics of HTTP operations make semantics useful to clarify the particular action being performed on an HTTP interaction. Discovering the precise semantics behind a service is a challenge that is most times achieved by sharing a semantic description of the interface of the service. Due to the bootstrapping problem of the Semantic Web, the thesis researches techniques to discover the services and automatically discover service descriptions.

The dissertation is organized as follows. The state of the art regarding the topics of the thesis is described in chapter 2. Chapter 3 describes the unified discovery framework that is followed in the thesis. Chapter 4 details the content level of the discovery framework, while chapter 5 describes the service level of the framework. The evaluation of the framework and the discovery techniques is detailed in chapter 6. Finally, chapter 7 draws some conclusions and proposes possible future work.

Chapter 2

State of the art

This chapter covers the state of the art in the context of the thesis. The thesis targets discovery of contents and services. Therefore, techniques for extracting information from unstructured web resources, and the main approaches and standards for service description and discovery are described in this chapter.

2.1 Introduction

As introduced in chapter 1, the thesis addresses the problem of resource discovery. This involves solving several issues, from screen scraping to service description.

Unstructured REST resources might contain relevant contents for discovery, or be services subject to be discovered. Web mining [Kosala and Blockeel, 2000] and, more specifically, information extraction [Chang et al., 2006] are research fields that cover the problem of extracting information from unstructured resources. The term Screen Scraping is the one usually employed to represent the kind of problem of extracting data out of unstructured HTML pages.

In the case of service discovery, there are several approaches to service modelling. These service description frameworks allow describing services, publishing their descriptions, and performing service discovery. As will be seen in section 5.2.2, the thesis researches techniques for automatically building service descriptions through an approach called service probing. Therefore, the thesis does not focus on defining a new service description framework. However, a background on these kinds of frameworks is given for the reader's interest.

2.2 Screen scraping

Plenty of approaches have already dealt with the problem of extracting information out of web sites which do not publish metadata that describe them. Usually, web servers return HTML representations of the web resources they host. The problem of Screen Scraping attempts to locate and extract relevant information from these HTML representations. Whenever the HTML code does not contain indicators to identify the relevant information, it is considered an unstructured document that is subject to being screen-scraped.

The techniques for scraping unstructured HTML representations of web resources are listed in this section. They can be classified into supervised and unsupervised approaches, depending on the requirement of human supervision over some data sets before the execution of training algorithms.

The techniques behind these kinds of systems are summarized in this section.

2.2.1 Regular expressions

In order to extract a particular piece of information out of an HTML document, one alternative consists of considering the document as plain text and applying

Document	Regular expression

`<html>`
`<body>`
`<p>Data: <b>32</b>`
`</p>`
`</body>`
`</html>`
	`<p>Data: <b>(.*)</b>`

`<html>`
`<body>`
`<p>`
`Data: <b>32</b>`
`</p>`
`</body>`
`</html>`
	`<p>\nData: <b>(.*)</b>`

Table 2.1: Regular expressions for two equivalent HTML documents

patterns for data extraction. These patterns are usually represented by regular expressions, which allow defining anchor texts that delimit the targeted data and extract these data.

An example would be extracting the location data from an HTML fragment such as `<p>Location: <span class=‘‘city’’>Madrid</span></p>`. The location data could be extracted by using the regular expression `<span class=‘‘city’’>(.*)</span>` .

Regular expressions are a quick and simple way to extract information from web resources. However, as they dismiss the tree structure of HTML documents, they are not proper HTML document parsers. The main disadvantage of this fact is the poor resistance to changes in serializations of the HTML document. An example of this issue is shown in table 2.1, which shows two HTML documents that have equivalent Document Object Model (DOM) trees but different serializations. Each one requires a different regular expression to select their data, which makes it hard to define robust regular expressions. Obviously, a common regular expression that fits both documents can easily be defined, but the example points out the drawback of mixing parsing and extraction tasks into one same pattern.

2.2.2 Tree matching

By dismissing the structured nature of HTML documents, regular expressions are vulnerable to changes in document serializations and are hard to maintain and generate. Unlike regular expressions, using the DOM tree of an HTML document

prevents parsing problems and allows operations such as tree matching. Figure 2.1 shows the DOM tree associated to an HTML document.

HTML

```
<html>
  <head>
  <title>Newspaper</title>
  </head>
  <body>
  <div class="header">
    <h1>
      Microsoft acquires Skype
    </h1>
    by <a href=
    "http://johntheblogger.com">
      John the blogger</a>
  </div>
  <div class="description">
    <p>After weeks of negotiation,
Microsoft has finally reached an
agreement for buying Skype.</p>
  </div>
  </body>
</html>
```

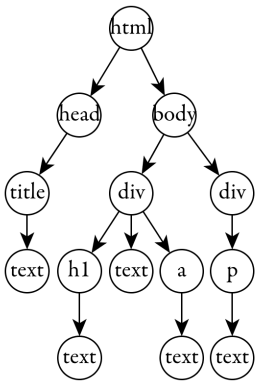

Figure 2.1: Example of DOM tree

An operation that is relevant to Screen Scraping is tree matching. Given two trees, tree matching is the set of mappings between the nodes of the trees. Tree matching is a method that is useful to induct tree patterns in web resources. It is based on tree edit distance, i.e. the minimum amount of editions that are needed to make on a tree to transform it into another one. Addition, removal or modification of a node are the considered possible editions.

An example of tree matching is shown in figure 2.2. In that figure, two different trees are shown with their respective mappings. The tree distance is two, as long as two operations (two node additions) are needed to transform one tree into another.

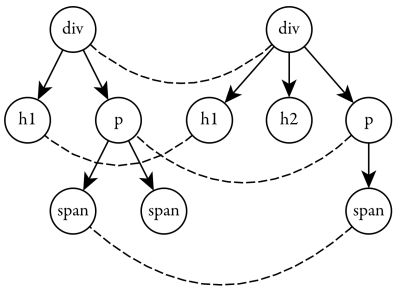

Figure 2.2: Example of tree matching

The matching among more than two trees is employed when trying to find a general pattern across many tree fragments and is called multiple tree alignment. The center star method [Gusfield, 1997] or partial tree alignment [Zhai and Liu, 2005a] are algorithms to perform this task, whose optimal solution has exponential time complexity [Carrillo and Lipman, 1988]. Therefore, given a set of similar tree fragments, it is possible to extract a general pattern that fits the samples. An example of multiple tree alignment with three trees is shown in figure 2.3, where an pattern tree is obtained after the alignment. In such figure, nodes marked with a ? symbol are optional in the pattern, whereas nodes marked with a * symbol might appear repeated times (from zero to any).

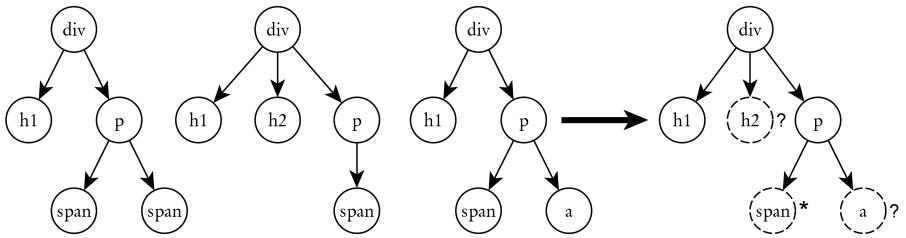

Figure 2.3: Example of multiple tree alignment

As will be seen, supervised approaches employ tree matching to induct generalized trees by processing tree fragments that have been previously selected and annotated by human users. Additionally, unsupervised approaches use tree edit distance to find similar fragments in web resources and automatically induct wrappers out of unannotated instances.

2.2.3 Wrapper induction

Wrapper induction is the technique in which, by using tree matching, inducts generalized patterns for data extraction. In such approach, given a target web page that wants to be scraped, a user manually annotates desired information that he/she wants to extract. With these annotated samples, a generalized pattern is inducted by using tree matching. Inducting a pattern ensures that if the data changes in the web resource, it will be extracted anyway, as the extractor is not tied to some specific data.

However, if the layout of the document changes in a way that the tree pattern does not fit it anymore, this technique would fail to extract the desired data. These

kinds of layout changes typically take place in web redesigns. Therefore, the vulnerability to layout changes is the major drawback of the technique of wrapper induction.

Some systems provide tools that allow the manual annotation of web resources and induct wrappers for information extraction. Some examples are Piggy Bank [Huynh et al., 2007], Reform [Toomim et al., 2009], Thresher [Hogue, 2005] and Marmite [Wong and Hong, 2007]. Piggy Bank provides also a web scraper that produces RDF out of web pages. Their approach is based on a browser plugin that provides a visual interface for programming the scraper and produces JavaScript as a result. Reform [Toomim et al., 2009] proposes scrapers that can be attached to web pages by non-programmers. The system is also integrated in Firefox web browser and allows users to annotate which parts of the web page contains a predefined pattern, and includes machine learning techniques for generalizing these annotations in the scraper. Thresher [Hogue, 2005] provides an interface where users can specify examples of semantic patterns by highlighting and assigning relevant features in the semantic web browser Haystack.

Also, some systems are targeted to experienced developers that need to construct scrapers and do not mind dealing with low-level concepts such as HTML, selectors, and wrappers. This is the case of systems like Chickenfoot [Bolin et al., 2005] or Denodo [Pan et al., 2002]. Both define JavaScript libraries for scraping, so that programmers can quickly and easily create extractors.

2.2.4 Vision-based approaches

Similarly, the systems that are based on visual features do not require manual supervision when applied to new web sites. Some approaches [Cai et al., 2003] such as the Vision-based Page Segmentation (VIPS) algorithm [Wei et al., 2006] perform this by defining some heuristics that are based on the HTML tags used for layouts, such as tables or titles. This limits the approach to the web sites that follow these design guidelines. Finally, another tool [Pembe and Güngör, 2010] uses visual information to extract data from web pages, which makes it work in web pages without preliminar knowledge about them. This tool only builds a hierarchical structure of the web page based on titles and sections.

There are studies that provide solutions to the problem of wrapper maintenance [Raposo et al., 2005], some of them using machine learning techniques for this purpose [Lerman et al., 2003]. However, they do not address the problem of wrapper generalization to different web sites.

2.2.5 GRDDL

Gleaning Resource Descriptions from Dialects of Languages (GRDDL) [Connolly, 2007] is the standard recommendation for constructing RDF graphs out of HTML or Extensible Markup Language (XML) documents. In order to extract the RDF semantic information, a GRDDL-aware agent should read the GRDDL annotations of the HTML/HTML resources, which are usually provided as Extensible Stylesheet Language Transformation (XSLT) annotations. GRDDL proposes a syntactic approach that can benefit from the reuse of scrapers among web sites. However, GRDDL specifies an unidirectional mapping between the implementation (XSLT scraper) and the web resource. This prevents this solution from dereferencing scraped data and reasoning about the scraping process. A transparent programmatic description would allow enhacements in the scraping process, such as reasoning on visual aspects of the web resource, distributed scraping using multiagent systems, or focused scraping and reasoning on the scraped resources (for example, search a piece of sports news about Rafa Nadal taken in China).

2.2.6 Unsupervised approaches

Some approaches perform automatic unsupervised wrapper induction. These systems attempt to identify repetitive patterns that occur in a document to build extraction patterns. This technique is applied by different extractors [Crescenzi et al., 2001] [Arasu, 2003] by identifying differences among neighbour resources, as well as in a supervised basis [Zhai and Liu, 2005b].

These approaches do not rely on user-annotated document fragments, and therefore require no interaction from a user. Some approaches integrate external knowledge to annotate the data type, such as Wikipedia [Cimiano et al., 2004], to classify and identify the type of the extracted structured data.

The main limitation with these approaches is that the output of the scraping is usually syntactical. Identifying the semantic relations of the extracted contents is out of their scope, and they focus on building a structure out of the unstructured input HTML document.

2.2.7 Discussion

This section discusses the techniques on Screen Scraping after covering the state of the art behind it. Some pending challenges are identified in the state of the art: robustness, generalization, and introspection.

The mentioned tools are not compatible among them. In fact, many scrapers are based on JavaScript code, leaving room for improvements if the extractor definitions were transparent. First, the definition of a general model for scrapers allows separating its programming model for its interface. In addition, scraping task goals could be queried, analysed and composed in a common language such as RDF. This is not feasible if these goals are specified in an implementation language such as JavaScript. These tools work on web sites that have been annotated by a user. The main limitation is that they require user supervision for every new web site, as long as a wrapper that was built for one web site cannot be applied to a different site.

2.3 Service discovery

The current web has applications with plenty of services available, which are of many different kinds. Approaches to describe services semantically allow automatic agents to execute and compose services automatically. The idea behind service discovery is that automatic agents are able to understand a service and its semantics in order to perform automatic execution and composition. The common approach to fulfil this goal is to define a service description that provides this information to the agent.

Service discovery involves identifying services that satisfy some specific requirements. In the context of the web, this involves tasks such as advertisement, storage, or matchmaking [Kirchberg et al., 2010].

Services are described in order to being advertised to service consumers. These service descriptions are usually documents in a standard format that can be processed by an automated processed to register the service or, in the case of semantic descriptions, perform advanced operations such as automatic composition.

Some approaches to semantic service description follow the Web Services Architecture, such as OWL Services (OWL-S) [World Wide Web Consortium, 2004], WSMO [Roman et al., 2005], or Semantic Annotations for WSDL (SAWSDL) [Kopecký et al., 2007]. To favour the integration with Internet's architecture, RESTful services started to be employed in web applications, which caused RESTful and lightweight alternatives to semantic web services to appear. SA-REST [Sheth et al., 2007] or hRESTs [Wright State University, 2008] are approaches that provide languages to describe RESTful APIs specifications. Similarly, WADL [Hadley, 2009] proposes describing RESTful Application Programming Inter-

faces (APIs) by defining a WADL, XML-based file. Also, [Vitvar et al., 2007] is a reduced variant of WSMO to simplify service description.

The type of service description determines the discovery process by affecting mainly the quality of the service matching. Service matching considers several techniques, which can be summarized into functional-based and non-functional-based methods [D'Mello and Ananthanarayana, 2010]. As functional-based methods the following approaches are proposed in the literature:

- Syntactic matching. Under this approach, lightweight techniques such as keyword/category matching [Curbera et al., 2002] or interface matching are employed [Wang and Stroulia, 2003]. In these cases, the matching only involves textual matching or comparison of input and output names or types, and no advanced semantic matching is performed.

- Behaviour-based matching [Park et al., 2009]. This approach focuses on the discovery of non-atomic services which are composition of other services. It works under the assumption that services are formally described, and works with process constraints and process algebras.

- Semantic matching. In this case, semantic information in the service description is employed to perform an advanced matching between the user's goals and the service function. The matching between the searched term and the candidate term employs semantic techniques like semantic distance or similar. For this, information retrieval methods [Wu and Chang, 2007], functional semantics-based matching [Ye and Zhang, 2006], ontology-based matching [Ji, 2009, Zhang and Li, 2005], Inputs Outputs Preconditions and Effects (IOPE)-based matching [Spanoudakis et al., 2007], and context information-based methods [Martin et al., 2005], are considered approaches.

Similarly, non-functional properties of services can be employed to perform the service matching. The approaches that are considered in the literature are Quality of Service (QoS) matching [Ye et al., 2009], usability-based matching [Namgoong et al., 2006], usage-based discovery [Birukou et al., 2007], or preferences-based methods [M'Bareck and Tata, 2007].

This section summarizes the main standards regarding service discovery, grouped into Web Services, Semantic Web Services, and Semantic REST services.

2.3.1 Web services

The Web Services architecture comprises a set of standards often known as WS-* standards. These standards provide means to model services syntactically in terms of messages, operations, and ports. The available operations are mapped or bound to some specific HTTP methods, Uniform Resource Locators (URLs) and parameters. This modelling freedom is a mismatch with the web's architectural style, where the uniform interface suggests using specific HTTP methods for each operation. These kinds of mismatches are the reason why the alternative of REST services has gained advocates recently. The section 2.3.3 will cover the alternative of REST services in contrast to Web Services. The current section reviews the main standards that comprise the Web Services stack in relation to service discovery: namely, Web Service Definition Language (WSDL), a service description language, and Universal Description, Discovery and Integration (UDDI), an architecture of service registry.

WSDL

WSDL [Christensen et al., 2001] is the language employed in the WS-* standards for describing services. It is an XML-based language which provides means to define messages, operations, and ports. WSDL only provides means to define a syntactical functional description, i.e. a functional description where only primitive types such as string, integer or boolean are allowed ways to model inputs, outputs or message types.

An example of WSDL document is the following [Christensen et al., 2001]:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:description
  targetNamespace="http://example.org/TicketAgent.wsdl20"
  xmlns:xsTicketAgent="http://example.org/TicketAgent.xsd"
  xmlns:wsdl="http://www.w3.org/ns/wsdl"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.w3.org/ns/wsdl http://www.w3.org/2007/06/wsdl/wsdl20.xsd">

  <wsdl:types>
    <xs:import schemaLocation="TicketAgent.xsd"
               namespace="http://example.org/TicketAgent.xsd" />
  </wsdl:types>

  <wsdl:interface name="TicketAgent">
    <wsdl:operation name="listFlights"
```

```
                pattern="http://www.w3.org/ns/wsdl/in-out">
      <wsdl:input element="xsTicketAgent:listFlightsRequest"/>
      <wsdl:output element="xsTicketAgent:listFlightsResponse"/>
    </wsdl:operation>

    <wsdl:operation name="reserveFlight"
                pattern="http://www.w3.org/ns/wsdl/in-out">
      <wsdl:input element="xsTicketAgent:reserveFlightRequest"/>
      <wsdl:output element="xsTicketAgent:reserveFlightResponse"/>
    </wsdl:operation>
  </wsdl:interface>
</wsdl:description>
```

As it can be observed in the example, the WSDL description provides information on the operation that the service performs, their input and output types and the endpoint (i.e. URL) used by the service.

The operation represents an exchange of messages. WSDL allows different patterns of message exchanges, to indicate whether a communication is unidirectional from client to server or viceversa, or bidirectional. It is also possible to indicate that an operation returns an answer only in case of failure.

The format used by the service messages is specified by using bindings. A WSDL service can employ HTTP for exchanging messages or encapsulate them into the Simple Object Access Protocol (SOAP) protocol (and in this case using HTTP as transport protocol and violating the REST architectural style constraints). WSDL bindings define the serialization details of the messages exchanged between client and server.

UDDI

UDDI is an architecture for service registry that can be employed by service providers to publish their services, and by consumers to query and retrieve services. A UDDI acts thus as a broker between service providers and consumers by publishing available WSDL service descriptions, enabling service discovery.

UDDI also provides services to describe businesses that provide services in the registry. A UDDI business registration includes white pages (contact information), yellow pages (industrial categorization based on taxonomies), and green pages (technical information about the services provided by the company).

At design time, UDDI was envisioned in a scenario where consumers would be linked to services by this kind of brokerage system. Service consumers would select services dynamically and at runtime. However, in practice the industry has not typically needed these features, which has resulted in alternative non-standard

registries to be proposed, such as Electronic Business using XML (ebXML)[1], Websphere Service Registry and Repository (WSRR) [2], or Mule Enterprise Service Bus [3].

2.3.2 Semantic Web Services

This section covers the approaches to attempt to achieve semantic service discovery by defining semantic service descriptions that automatic agents can process and use. The difference with traditional Web Services is the level of automation that semantics allow. Web Services descriptions such as WSDL only cover the syntactic level of service modelling.

For example, the interface needed to execute a service is defined in terms of parameter names or input types, which allows a human administrator to get a certain idea about how to use the service. This human administrator relies on his natural language knowledge to properly interpret the parameter names and other textual descriptions. An automatic process, however, is not capable of taking advantage of such descriptions to the same level.

By using semantics to represent the elements that describe services, automatic agents are allowed to employ relations among similar concepts for tasks such as service selection and mediation, among others. This allows automating the development of composed applications by enabling automatic selection, execution and composition.

The main approaches to build semantic service descriptions are covered next.

OWL-S

OWL-S is the standard for describing Semantic Web Services that is based on Web Ontology Language (OWL). Because of being based on OWL, it fits naturally other OWL ontologies, provinding an OWL vocabulary for service description. The motivating tasks for OWL-S are enabling automatic web service discovery, automatic invocation, and automatic composition and interoperation. OWL-S attempts to allow declarative advertisements of service properties and capabilities that can be used for automatic service discovery.

There are four main elements in OWL-S descriptions, as shown in figure 2.4. OWL-S acts as an upper ontology for services. The service class shown in the

[1] `http://www.ebxml.org/`
[2] `http://www-01.ibm.com/software/integration/wsrr/`
[3] `http://www.mulesoft.org/`

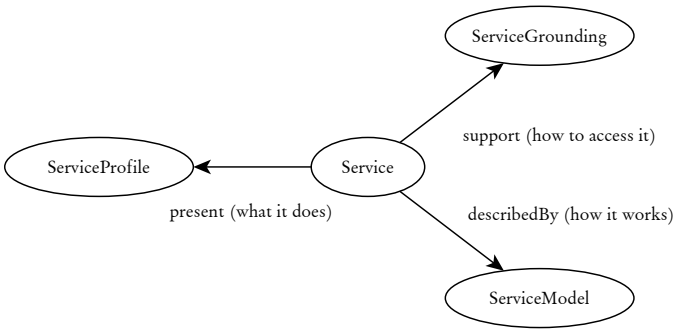

Figure 2.4: OWL-S elements

figure is extended with three subclasses, which attempt to answer three questions about the modelled service:

- What does the service provide for clients?. The service profile attempts to answer this question. In practice, an OWL-S subclass of service profile is defined and semantic matching is made against this concept at matchmaking time. Additionally, the profile allows defining information about the service provider, the function in terms of IOPEs, and non-functional properties such as service quality and ratings.

- How is it used?. This is given by the service process model. The semantic content of requests and responses, conditions under which certain outcomes will take place and possible step-by-step processes are defined by the service model. Therefore, services that involve several interactions on different methods are subject to have a service model that provides useful information to agents interested in using these services' functionalities.

- How does one interact with it?. The service grounding answers this question by allowing defining the usage of underlying transport protocols. Typically, the service grounding will be tied to a WSDL description by specifying a communication protocol, message formats, Uniform Resource Identifiers (URIs), and ports.

An example of OWL-S service description is shown next:

```
<rdf:RDF
  xmlns:rdf=     "&rdf;#"
  xmlns:rdfs=    "&rdfs;#"
```

```
  xmlns:owl = "&owl;#"
  xmlns:service= "&service;#"
  xmlns:process= "&process;#"
  xmlns:profile= "&profile;#"
  xmlns:actor= "&actor;#"
  xmlns:addParam= "&addParam;#"
  xmlns:profileHierarchy= "&profileHierarchy;#"
  xmlns:country= "&country;#"
  xmlns:concepts= "&concepts;#"
  xmlns:ba_process= "&ba_process;#"
  xmlns:ba_service= "&ba_service;#"
  xmlns=         "&DEFAULT;#"
  xml:base=      "&DEFAULT;">

  <profileHierarchy:AirlineTicketing rdf:ID="Profile_BravoAir_ReservationAgent">
    <service:presentedBy rdf:resource="&ba_service;#BravoAir_ReservationAgent"/>
    <profile:has_process rdf:resource="&ba_process;#BravoAir_Process"/>


    <profile:serviceName>BravoAir_ReservationAgent</profile:serviceName>
    <profile:textDescription>
This  service provide flight reservations based on the
specification of a flight request.  This typically involves a departure
airport, an arrival airport, a departure date, and if a return trip is
required, a return date.
        If the desired flight is available, an itinerary and reservation number
will be returned.
    </profile:textDescription>

    <profile:serviceParameter>
      <addParam:GeographicRadius rdf:ID="BravoAir-geographicRadius">
<profile:serviceParameterName>
  BravoAir Geographic Radius
</profile:serviceParameterName>
<profile:sParameter rdf:resource="&country;#UnitedStates"/>
      </addParam:GeographicRadius>
    </profile:serviceParameter>

    <profile:hasInput rdf:resource="&ba_process;#DepartureAirport"/>
    <profile:hasInput rdf:resource="&ba_process;#ArrivalAirport"/>
    <profile:hasInput rdf:resource="&ba_process;#OutboundDate"/>
    <profile:hasInput rdf:resource="&ba_process;#InboundDate"/>
    <profile:hasInput rdf:resource="&ba_process;#RoundTrip"/>
    <profile:hasInput rdf:resource="&ba_process;#AcctName"/>
    <profile:hasInput rdf:resource="&ba_process;#Password"/>
    <profile:hasInput rdf:resource="&ba_process;#Confirm"/>
    <profile:hasOutput rdf:resource="&ba_process;#FlightsFound"/>
    <profile:hasOutput rdf:resource="&ba_process;#PreferredFlightItinerary"/>
    <profile:hasOutput rdf:resource="&ba_process;#ReservationID"/>
    <profile:hasResult rdf:resource="&ba_process;#HaveSeatResult"/>
```

```
  </profileHierarchy:AirlineTicketing>
</rdf:RDF>
```

Regarding the annotation task, it is worth mentioning the METEOR-S framework [Patil et al., 2004, Sivashanmugam et al., 2003], which helps in annotating WSDL descriptions with terms from OWL-S ontology. This framework was later on extended to support other description models such as WSMO.

Therefore, an OWL-S description captures information about inputs, outputs, preconditions, and effects, all grouped into an OWL class that can be matched using semantic technologies. Regarding service matchmaking, four types of matching are typically distinguished when analyzing the matching of a desired service (i.e. a user's goal) with actual described services [Paolucci et al., 2002]:

- Exact match. The output of the request and the one of the service are exactly the same.
- Plugin match. The output of the service subsumes the output of the request.
- Subsumption match. The output of the request subsumes the output of the service.
- Fail. There is no matching between the compared service profiles.

Additionally, process information is provided so that step-by-step interactions (e.g., booking a ticket, which usually involves a search task and a booking task) can be known in advance by automatic agents. Finally, the necessary means to execute the service is present thanks to the service grounding.

WSMO

WSMO [ESSI WSMO working group, 2004] is an initiative for semantic web service description that proposes a framework and a set of ontologies for such purpose. It is based on the following principles:

- Web compliance. WSMO identifies resources using URIs and adopts namespaces for defining information spaces. Additionally, it supports XML and other World Wide Web Consortium (W3C) recommendations like resource decentralization. This does not necessarily mean that WSMO sticks to all guidelines behind the REST architectural style principles.

- Ontology-based. Resource descriptions and all interchanged data by services are represented using ontologies. Although WSMO thus supports Semantic Web vision, it uses alternative representation languages for its ontologies, instead of RDF and OWL standards.
- Strict decoupling. WSMO resources are defined in isolation. I.e. resources are independent and without being specified to perform an implicit usage of other resources.
- Centrality of mediation. WSMO addresses the problem of mediation in service composition and makes it a core component of the framework in order to tackle heterogeneity problems on data, protocols or processes.
- Ontological role separation. WSMO makes a difference between user desires and the services available in the system, as long as users' contexts are usually not the same as the ones the web services have been designed in.
- Description versus implementation. WSMO separates the description of service elements from the actual executable technologies employed. It focuses on providing a description model without concerning about the implementation that allows the web services to be executed.
- Execution Semantics. In order to verify the WSMO specification, the formal execution semantics of reference implementations like Web Service Modelling Execution Environment (WSMX) as well as other WSMO-enabled systems provide the technical realization of WSMO.
- Service versus web service. WSMO differentiates the two concepts by considering a web service a computational entity that is able to achieve a user goal, while a service is the actual value provided by the invocation [Preist, 2004]. Therefore, in WSMO web services provide access to services.

WSMO defines a formal model for describing services. Service descriptions consist of preconditions and postconditions that model the service effects on the inputs, outputs, and the system's state. An example of a WSMO description of a ticket booking web service is shown next [Feier and Domingue, 2005]:

```
namespace {_"http://example.org/bookTicket#",
   dc       _"http://purl.org/dc/elements/1.1#",
   tr        _"http://example.org/tripReservationOntology#",
   foaf     _"http://xmlns.com/foaf/0.1/",
```

```
   wsml   _"http://www.wsmo.org/wsml/wsml-syntax#",
   bti      _"http://www.example.org/BookTicketInterfaceOntology#"}
webService _"http://example.org/bookTicketWebService"
importsOntology _"http://example.org/tripReservationOntology"
capability BookTicketCapability
interface BookTicketInterface

capability BookTicketCapability
  sharedVariables {?creditCard, ?initialBalance, ?trip,
                   ?reservationHolder, ?ticket}
  precondition
      nonFunctionalProperties
           dc#description hasValue "The information of a trip which starts in
            Austria together with the information about the person who wants to
            have a reservation must be given as a part of a reservation request.
            A credit card is also required."
      endNonFunctionalProperties
      definedBy
          ?reservationRequest[
              reservationItem hasValue ?trip,
              reservationHolder hasValue ?reservationHolder
           ] memberOf tr#reservationRequest
           and
           ?trip memberOf tr#tripFromAustria and
           ?creditCard[
              balance hasValue ?initialBalance
           ] memberOf po#creditCard.
  assumption
      nonFunctionalProperties
          dc#description hasValue "The credit card information provided by the
           requester must designate a valid credit card that should be either
           PlasticBuy or GoldenCard."
      endNonFunctionalProperties
      definedBy
           po#validCreditCard(?creditCard)
           and ( ?creditCard[
                        type hasValue "PlasticBuy"]
                  or
                 ?creditCard[
                        type hasValue "GoldenCard"]
                ).
  postcondition
      nonFunctionalProperties
          dc#description hasValue "A reservation containing the details of a ticket
           for the desired trip and the reservation holder is the result of the
           successful execution of the Web service."
      endNonFunctionalProperties
      definedBy
         ?reservation memberOf tr#reservation[
            reservationItem hasValue ?ticket,
            reservationHolder hasValue ?reservationHolder
```

```
        ]
        and
        ?ticket[
           trip hasValue ?trip
        ]
        memberOf tr#ticket.
effect
   nonFunctionalProperties
        dc#description hasValue "The credit card  will be charged with the cost of
         the ticket."
   endNonFunctionalProperties
    definedBy
        ticketPrice(?ticket, "euro", ?ticketPrice)
        and
       ?finalBalance= (?initialBalance - ?ticketPrice)
        and
        ?creditCard[
             po#balance  hasValue  ?finalBalance
         ] .
```

Regarding service discovery, WSMO describes three steps in the process of discovery: goal discovery, web service discovery, and service discovery. As said, according to WSMO terminology, a web service is a computational entity, while a service is the actual obtained value. Goal discovery targets retrieving abstract goal descriptions given some inputs that are provided by a user, such as input keywords or logical expressions. Web service discovery attempts to identify suitable abstract web services that match the discovered goals. Finally, service discovery targets the discovery of actual services that fit the abstract web service description previously discovered.

For the service discovery task, three approaches, varying in complexity, are considered: syntactical matching, lightweight semantic matching, and heavyweight semantic matching. Syntactical approaches do not vary greatly from WSDL-based discovery and are based on keyword-based search or basic Natural Language Processing (NLP) methods. Lightweight semantic matching considers ontologies, action-object-modelling, and coarse-grained semantic descriptions, very much in the fashion of the already mentioned OWL-S matchmaking (exact matching, plugin matching, subsumption matching, etc.). Finally, heavyweight semantic matching considers detailed service descriptions, with description of capabilities and states.

SWSF

Semantic Web Services Framework (SWSF) [Battle et al., 2005] is an approach to semantic service description that proposes an ontology called Semantic Web Services Ontology (SWSO) and modelling language Semantic Web Services Language (SWSL) that is employed to define formal characterizations of services. SWSF proposes two variants for service modelling: using first-order logic or logic programming, resulting in two different ontologies: First-order Logic Ontology for Web Services (FLOWS) and Rule Ontology for Web Services (ROWS), respectively.

SWSF is similar to OWL-S due to the fact that SWSF proposes three views of a service: service descriptor, process model, and grounding (in the way of OWL-S service profile, service process and service grounding). The main difference resides in the expressiveness of the language employed by SWSF, which is based on first-order logic, making use of logic predicates and terms to model states. Aspects such as terms which vary over time are used to model changes in the world.

SAWSDL

SAWSDL [Kopecký et al., 2007] allows mapping a syntactical functional description in WSDL to a semantic service description such as OWL-S or WSMO. SAWSDL specification is motivated by the continuous growth in service description frameworks. Instead of defining a language for representing the semantic models, SAWSDL provides mechanisms to reference semantic entities from within WSDL annotations.

Although SAWSDL does not define a service model, it introduces some terminology of interest:

- Semantic model. A set of representations that are used to model an area of knowledge. Ontologies are an example of semantic models.
- Concept. Any element of a semantic model, identified by a URI.
- Semantic annotation. Additional information in a document's element that maps such element to a concept of a semantic model.
- Semantics. The set of concepts identified by semantic annotations.

SAWSDL is defined by the principle of enhancing WSDL with annotations that are agnostic to the semantic model employed. An example of a semantically-annotated WSDL using SAWSDL is given next:

```
<wsdl:description
  targetNamespace="http://www.w3.org/2002/ws/sawsdl/spec/wsdl/order#"
  xmlns="http://www.w3.org/2002/ws/sawsdl/spec/wsdl/order#"
  xmlns:wsdl="http://www.w3.org/ns/wsdl"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:sawsdl="http://www.w3.org/ns/sawsdl">

  <wsdl:types>
    <xs:schema targetNamespace="http://www.w3.org/2002/ws/sawsdl/spec/wsdl/order#"
      elementFormDefault="qualified">
      <xs:element name="OrderRequest"
          sawsdl:modelReference=
            "http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder#OrderRequest"
          sawsdl:loweringSchemaMapping=
            "http://www.w3.org/2002/ws/sawsdl/spec/mapping/RDFOnt2Request.xml">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="customerNo" type="xs:integer" />
            <xs:element name="orderItem" type="item" minOccurs="1"
                        maxOccurs="unbounded" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:complexType name="item">
        <xs:all>
          <xs:element name="UPC" type="xs:string" />
        </xs:all>
        <xs:attribute name="quantity" type="xs:integer" />
      </xs:complexType>
      <xs:element name="OrderResponse" type="confirmation" />
      <xs:simpleType name="confirmation"
          sawsdl:modelReference=
      "http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder#OrderConfirmation">
        <xs:restriction base="xs:string">
          <xs:enumeration value="Confirmed" />
          <xs:enumeration value="Pending" />
          <xs:enumeration value="Rejected" />
        </xs:restriction>
      </xs:simpleType>
    </xs:schema>
  </wsdl:types>

  <wsdl:interface name="Order"
      sawsdl:modelReference=
        "http://example.org/categorization/products/electronics">
    <wsdl:operation name="order" pattern="http://www.w3.org/ns/wsdl/in-out"
        sawsdl:modelReference=
"http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder#RequestPurchaseOrder">
      <wsdl:input element="OrderRequest" />
      <wsdl:output element="OrderResponse" />
    </wsdl:operation>
```

```
  </wsdl:interface>
</wsdl:description>
```

As it can be observed, a plain WSDL document is enriched with semantic annotations. The construct `modelReference` is employed to indicate a one-to-one mapping between WSDL elements and semantic concepts. Similarly, the construct `schemaMapping` allows associating XML Schema Definition (XSD) elements to semantic data as one-to-many mappings.

USDL

Unified Service Description Language (USDL) [Kadner and Oberle, 2011] is a language for describing services. Its follows a more general approach than other frameworks in the sense that it targets a broad definition of service, and not just web services. Examples of services that are targeted by USDL are [SAP Research, 2011]:

- Purely human/professional: e.g., project management and consultancy.
- Transactional: e.g., purchase order requisition.
- Informational: e.g., spatial and demography look-ups.
- Software component: e.g., software widgets for download.
- Digital media: e.g., video and audio clips.
- Platform: e.g., middleware services such as message store-forward.
- Infrastructure: e.g., CPU and storage services.

Such generic approach to service description makes that not all aspects of USDL are applied to all domains. USDL needs to be tailored for the specific needs of applications, where concepts are adapted and new ones introduced. USDL takes initiatives such as WSDL as starting point and adds business and operational information on top. To achieve this, Modules for pricing, legal, functional, participants, interactions and Service-Level Agreement (SLA) aspects are defined to extend the service description.

2.3.3 Semantic REST services

In contrast with the Web Services architecture, REST services attempt to fit in a better way the architectural style of the web. Web Services couple applications they integrate and only use the web as a transport platform, incurring into mismatches by violating architectural constraints of REST [Pautasso et al., 2008]. The main differences of REST services in comparison to traditional Web Services are the following:

- Resource identification through URIs. Every resource in the web has, by definition, a URI. REST services are web resources (or a set of them) and therefore their execution endpoints are URIs.

- Uniform interface. Unlike Web Services, REST services are simply web resources whose possible interactions should be predictable depending on the HTTP method employed in the HTTP interaction. E.g. `get` method is used to retrieve the representation of a resource, while `PUT` is employed to update a resource. The semantics of each HTTP method are the same for every web resource.

- Self-descriptive messages. Resources cannot be accessed, but only their representations. Additionally, these representations can be retrieved in a variety of formats, which can be negotiated between client and server using HTTP request headers.

- Stateful interactions through hyperlinks. As long as interactions with REST resources are stateless because of architectural decision, states need to be maintained using techniques such as cookies, session-identified URIs or hidden form fields.

The number of architectural constraints behind REST is bigger than the Web Services architecture. Additionally, the pervasiveness of web tools such as web browsers and the fact that REST follows well-known standards such as HTTP, XML, URI, or Multipurpose Internet Mail Extensions (MIME) make that REST services are usually perceived to be simpler than the Web Services stack. This makes REST an approach recently regarded as more elegant for application composition than the Web Services architecture. This section reviews the approaches to provide semantic service descriptions to REST services.

WADL

WADL [Hadley, 2009] defines a format for building and publishing RESTful semantic service descriptions which can be discovered and processed by automatic agents. It attempts to follow an approach similar to WSDL but applied to REST services. Hence, a document that defines syntactic aspects of the service is published in order to be discovered by automatic processes.

An example of WADL document is given next [Hadley, 2009]:

```
<?xml version="1.0"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://wadl.dev.java.net/2009/02/wadl.xsd"
 xmlns:tns="urn:yahoo:yn"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:yn="urn:yahoo:yn"
 xmlns:ya="urn:yahoo:api"
 xmlns="http://wadl.dev.java.net/2009/02">
  <grammars>
    <include
      href="NewsSearchResponse.xsd"/>
    <include
      href="Error.xsd"/>
  </grammars>
  <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
    <resource path="newsSearch">
      <method name="GET" id="search">
        <request>
          <param name="appid" type="xsd:string"
            style="query" required="true"/>
          <param name="query" type="xsd:string"
            style="query" required="true"/>
          <param name="type" style="query" default="all">
            <option value="all"/>
            <option value="any"/>
            <option value="phrase"/>
          </param>
          <param name="results" style="query" type="xsd:int" default="10"/>
          <param name="start" style="query" type="xsd:int" default="1"/>
          <param name="sort" style="query" default="rank">
            <option value="rank"/>
            <option value="date"/>
          </param>
          <param name="language" style="query" type="xsd:string"/>
        </request>
        <response status="200">
          <representation mediaType="application/xml"
            element="yn:ResultSet"/>
         </response>
        <response status="400">
```

```
          <representation mediaType="application/xml"
            element="ya:Error"/>
        </response>
      </method>
    </resource>
  </resources>
</application>
```

As can be observed, a service is modelled as a set of resources. Each resource contains a set of methods that describe the possible interactions with that particular resource. For each method, requests and responses are modelled. A request is modelled as a set of input parameters. Input parameters have a name and a type, and might have a closed set of allowed values. Also, a method includes the available responses, which have a type, a MIME type, and a status code. Requests can also specify any needed HTTP header, while responses allow dealing the errors and failures during the service execution.

WADL descriptions are syntactic in the sense that no semantic information is provided, but just primitive types of input parameters. This allows some basic code generation functionalities client-side, but no advanced discovery or selection. Input parameters are identified by plain text names that does not convey their semantics, thus not allowing mediation for composition. On the other hand, WADL draws concepts from WSDL and applies them to the case of REST services, where different HTTP methods are employed and non-SOAP responses are allowed.

MicroWSMO

MicroWSMO is a lightweight revision of WSMO that can be employed to annotate services directly on HTML pages. Typically, services are documented in web sites that provide textual descriptions and examples of the services. MicroWSMO proposes reusing these documentation pages for building service descriptions by including semantic annotations directly into the HTML documents. For such task, MicroWSMO uses the reduced revision of WSMO, called WSMO-Lite. The model behind WSMO-Lite allows describing both RESTful and RESTless web services in a similar though more lightweight fashion as the full-fledged WSMO standard. WSMO-Lite does not include goals and mediators, present in WSMO. Besides, WSMO-Lite uses a simplified model for IOPEs, grouping them into preconditions and effects, and, unlike WSMO, allows categorizing services using taxonomies.

An example of annotated HTML document by using MicroWSMO is shown

next:

```
<div class="service" id="svc">
  <h1><span class="label">ACME Hotels</span> service API</h1>
  <p>This service is a
    <a rel="model" href="http://example.com/ecommerce/hotelReservation">
    hotel reservation</a> service.
  </p>
  <div class="operation" id="op1">
    <h2>Operation <code class="label">getHotelDetails</code></h2>
    <p>
      Invoked using the <span class="method">GET</span>
      at <code class="address">http://example.com/h/{id}</code><br/>
      <span class="input">
        <strong>Parameters:</strong>
        <a rel="model" href="http://example.com/data/onto.owl#Hotel">
        <code>id</code></a> - the identifier of the particular hotel
        (<a rel="lowering" href="http://example.com/data/hotel.xsparql">lowering</a>)
      </span><br/>
      <span class="output">
        <strong>Output value:</strong> hotel details in an
        <code>ex:hotelInformation</code> document
      </span>
    </p>
  </div>
</div>
```

As it can be observed in the sample fragment, there are no RDFa annotations, thus no RDF schema is strictly present in the HTML document. Instead, MicroWSMO relies on using Poshformats (i.e. non-standard Microformats, or the embedment of particular HTML classes with agreed semantics) for representing the RDF relations, extending the hRESTS framework, which is detailed later, with WSMO-Lite data.

A particularization of WSMO-Lite is Resource-Oriented Service Model (ROSM) [Fischer and Norton, 2009], a tailored version for REST services. ROSM is thus a WSMO subset that applies concepts from WSMO on a more restrictive and lightweight model that fits the REST architectural style. The defined model is similar to WADL, allowing the definition of input parameters and expected outputs, though RDF is used for the definition of the schema, as Semantic Web standard in line with Linked Data principles.

hRESTS and SA-REST

Other RESTful approaches are Semantically-Annotated REST (SA-REST) [Sheth et al., 2007] and hRESTS [Wright State University, 2008, Kopecky et al., 2008],

which allow building semantic service descriptions by annotating textual descriptions of services' APIs with respectively RDFa and Microformats [Microformats community, 2008].

An SA-REST-enhanced HTML page is a regular HTML document with RDFa annotations that provide the semantics of the service, as shown in the following example:

```
<html xmlns:sarest="http://lsdis.cs.uga.edu/SAREST#">
...
<p about=" http://craigslist.org/search/">
  The logical input of this service is an
  <span property="sarest:input">
    http://lsdis.cs.uga.edu/ont.owl#Location_Query
  </span>
  object. The logical output of this service is a list of
  <span property="sarest:output">
    http://lsdis.cs.uga.edu/ont.owl#Location
  </span>
  objects. This service should be invoked using an
  <span property="sarest:action">
    HTTP GET
  </span>
  <meta property="sarest:lifting"
        content="http://craigslist.org/api/lifting.xsl"/>
  <meta property="sarest:lowering"
        content="http://craigslist.org/api/lowering.xsl"/>
  <meta property="sarest:operation"
        content="http://lsdis.cs.uga.edu/ont.owl#Location_Search"/>
</p>
```

Similarly, hRESTS use Poshformats, or agreed semantics of HTML annotations, to annotate APIs documentations, as shown in the next example:

```
<div class="service" id="svc">
  <p>Description of the
  <span class="label">ACME Hotels</span> service:</p>
  <div class="operation" id="op1">
    <p>The operation <code class="label">getHotelDetails</code> is
      invoked using the method <span class="method">GET</span>
      at <code class="address">http://example.com/h/fidg</code>,
      with <span class="input">the ID of the particular hotel replacing
      the parameter <code>id</code></span>.
      It returns <span class="output">the hotel details in an
      <code>ex:hotelInformation</code> document.</span>
    </p>
  </div>
</div>
```

Both approaches use a similar service model. Essentially, they provide a schema to represent input and output messages on particular service operations that are tied to a URI. Also, the HTTP method used in the operation is defined as part of the model. These properties are included as HTML classes in hRESTS, and as RDFa properties in SA-REST.

RDForms

RDForms research focus on HTML form description. RDForms [Baker, 2005] attempt to "add to the Semantic Web capabilities similar to HTML forms". Schemas for indexable, container and settable operations are defined, which represent HTTP GET, POST, and PUT methods, respectively.

An example of RDForm is shown next[4]:

```
<form id="personform" action="" about="#form1" typeof="pb:RDForm">
    <div rel="pb:operation">
      <div about="#crud-op1" typeof="pb:CRUDOperationDELETE">
        <span rel="pb:onField" resource="#form1.field1" />
      </div>
    </div>
    <fieldset>
      <legend>Person</legend>
        <div rel="pb:field">
          <div about="#form1.field1" typeof="pb:UpdateableField">
            <label rel="pb:key" resource="#form1.field1.key">Name</label>:
            <div rel="pb:value">
              <div about="#form1.field1.value" typeof="pb:FieldValue">
                <input type="text" id="person-name" property="rdf:value"
                       content="Jason Dunno" value="Jason Dunno" />
              </div>
            </div>
          </div>
        </div>
    </fieldset>
</form>
```

RDForms allow defining interfaces to REST resources using RDF by enhancing the execution forms through the use of RDFa. The proposed model does not differ greatly from other RESTful approaches to service description. Hence, the proposed schema is similar to WADL or ROSM, though adding the idea of enriching HTML forms with semantic annotations.

[4] `http://www.w3.org/wiki/PushBackDataToLegacySourcesRDForms`

OpenSearch

On the specific topic of search services, OpenSearch [A9.com, inc., 2005] is an approach to the description of search services. By creating an OpenSearch description document, a search service is published in a similar way as other standards for service description are applied to services in general.

An example of OpenSearch description is shown next:

```
<?xml version="1.0" encoding="UTF-8"?>
  <OpenSearchDescription xmlns="http://a9.com/-/spec/opensearch/1.1/">
    <ShortName>Web Search</ShortName>
    <Description>Use Example.com to search the Web.</Description>
    <Tags>example web</Tags>
    <Contact>admin@example.com</Contact>
    <Url type="application/rss+xml"
         template=
           "http://example.com/?q={searchTerms}&amp;pw={startPage?}&amp;format=rss"/>
    <LongName>Example.com Web Search</LongName>
    <Image height="64" width="64" type="image/png">
      http://example.com/websearch.png</Image>
    <Query role="example" searchTerms="cat" />
    <Developer>Example.com Development Team</Developer>
    <Attribution>
      Search data Copyright 2005, Example.com, Inc., All Rights Reserved
    </Attribution>
    <SyndicationRight>open</SyndicationRight>
    <AdultContent>false</AdultContent>
    <Language>en-us</Language>
    <OutputEncoding>UTF-8</OutputEncoding>
    <InputEncoding>UTF-8</InputEncoding>
  </OpenSearchDescription>
```

The description document enumerates the services' capabilities and configurations. OpenSearch delegates to service providers to build an appropriate RESTful interface that can be described using OpenSearch. Some services might require adaptation in order to be described by an OpenSearch description, but this issue is not addressed. OpenSearch is limited to search services, so it considers fine-grain aspects such as content encoding, formatting, and type of results.

2.3.4 Other approaches

Many research fields have already researched service discovery [Etzioni, 1996] in parallel to the development of the web. Research behind Foundation for Intelligent Physical Agents (FIPA) agents has already analysed the problem of service discovery, developing several protocols for the diverse possible environments [Pirker

et al., 2004] [Jun et al., 2000] [Cao et al., 2001]. In pervasive computing, where services are widespread around *ad-hoc* networks, service discovery is a research field [Ratsimor et al., 2004] [Chakraborty et al., 2006]. These approaches differ greatly because of the RESTless nature of the platform they are based on, i.e. often middlewares of different kind. Some approaches [Sycara et al., 2004] [Colucci et al., 2004] [Sidnal et al., 2010] [Neiat et al., 2009] consider semantic web services, where the web is used more as platform by following the web services architectures instead of RESTful services. [Khriyenko and Nagy, 2011] considers the RESTful architecture of the Web and integrates an agent-based framework on it and the Linked Data initiative, though it does not consider the automatic construction of the semantic descriptions that are required for these kinds of solutions.

Some research approaches in the current literature also perform service mining tasks. Wang et al. [2011] mine Programmable Web and build a domain ontology out of the keywords available in the textual descriptions of the services. It helps to validate Programmable Web's categorization scheme. Blake and Nowlan [2011] performs an automatic categorization of services using the internals of WSDL descriptions, and not just the keywords available in the textual descriptions. Elmeleegy et al. [2008] is a mashup advisor, which also builds a catalogue of mashup components to exploit in recommendations for mashup development and ranks components for their use.

2.3.5 Discussion

This section has reviewed the state of the art in service discovery. To sum up, service discovery involves a set of tasks from service description to the actual selection of the service through service matching. For these tasks, there are several techniques that can be employed. In general, most service discovery solutions found in the literature follow a similar paradigm. A service description model is proposed, which is employed to build a service description that is advertised for service consumers to use it. Service registries aggregate descriptions and consumers query services according to some selection constraints to find a desired service. Whenever these service descriptions are not available, the chain breaks and no matching can be performed as long as no services are registered. Therefore, the challenge behind service discovery lies in integrating the task of service description to the process of service discovery. Hence, research on automated service description would lead to a service discovery process that does not rely on the effort of developers on describing services semantically.

2.4 Conclusions

As described, there are different approaches for information extraction in web resources depending on the manual or automatic method, the supervised or unsupervised approach, and the performance of each technique. The main challenges that have been identified deal with robustness, generalization and introspection capabilities of the approaches.

In the case of robustness, there are still many approaches that require a layout that is fixed in time whenever performing screen scraping. In other words, once a web resource's representation changes its HTML layout, many wrapper techniques fail to keep providing appropriate results. Similarly, generalization capabilities are still a problem, in the sense that most screen scraping techniques require that extractors are defined *ad-hoc* for each targeted web resource. An inducted wrapper for a web resource does not work for other web resources which provide the same types of contents, as long as HTML layouts are not the same [Kushmerick, 1997]. Finally, fragmentation in extractor techniques and tools lead to a non-standard representation of wrappers. As mentioned, some approaches even use programming languages such as JavaScript to represent wrappers. This makes it not possible to reason about extractors or progressively improve them, hampering the development of more advanced applications such as focused crawlers or agent-based scrapers.

Regarding services, several approaches deal with web components of different kinds, from services to widgets. There are many initiatives to describe services' interface to allow automation of certain tasks, in the Web Service field [Christensen et al., 2001] [ESSI WSMO working group, 2004], or in the REST service area with heavy-weight approaches such as WADL [Hadley, 2009], or more light-weight approaches such as WSMO-Lite [Vitvar et al., 2007], SA-REST [Sheth et al., 2007] or hRESTS [Wright State University, 2008]. W3C widgets [Alario-Hoyos and Wilson, 2010] define a standard for describing widgets. These and other already mentioned alternatives such as WSMO, SA-REST or WADL are heavy-weight approaches that provide means to describe every possible service from scratch.

The reviewed approaches for service discovery allow describing the inners of these components, and they operate at an abstraction level that is lower than the models proposed in the thesis. As will be seen in the thesis, they will be used by allowing linking Linked Mashups Ontology (LiMOn) or feature-oriented descriptions to WSDL/WSMO descriptions. In all of the reviewed approaches it

is expected that a developer provides a service description which one published makes the service discoverable, in contrast to this thesis' proposal, which attempts to build these kinds of descriptions to achieve discovery.

Chapter 3

Discovery framework

This chapter describes the integrated framework for content and service discovery and extraction. The framework is divided in several levels where discovery of contents and services is done in a RESTful system such as the Web. The lowest level is the content discovery level, where contents are extracted out of web pages. Rules are used in that level to map data from unstructured REST resources to actual semantic entities. The middle level is the service level, where discovery rules are employed to map discovered content from HTTP interactions to semantic service descriptions. The top level is the agent level, where orchestration takes place by the use of rules that act as plans for crawling and service execution.

3.1 Introduction

The phenomenon of Web 2.0 has caused the mushrooming of websites and applications that allow users producing content without any sort of technical skill. This has caused the web to be a highly rich source of contents in the so-called Digital Age thanks to users' collaboration and other content providers. Also, with the emergence of mashup technologies, developers are able to combine existing services and data sources to quickly build new applications, in a similar fashion to Web 2.0.

This vast amount of information and services available in the web makes it a platform for development of mashed-up applications with intensive information usage. The research field of Semantic Web [Berners-Lee et al., 2001] and the Linked Data initiative [Bizer et al., 2009] have defined techniques and standards for the semantic representation of information, which have been experiencing an increasing adoption in the web.

However, there are still plenty of web sites that do not provide appropriate semantic metadata in the resources they publish. This causes that their services and contents are only processable by the human users that visit these web sites. The reasons for this can be various: because of limited knowledge by developers about Semantic Web standards or because of limited effort that developers can spend on these tasks.

Also, the so-called deep web contains many web resources that are not discoverable by crawlers. The deep web is the subset of the web that is only accessible behind web forms. It thus requires a different treatment for reaching the information and accessing their contents and services, which makes this part of the web hidden to most automatic agents.

In this chapter, a framework for the discovery of services and contents in the web is proposed. The framework allows intelligent focused crawling [Chakrabarti and Dom, 1999] and agents accessing the deep web. Therefore, an agent architecture that uses this discovery framework for goal-oriented discovery of resources is also described. This agent architecture allows implementing agents that intelligently crawl and use services for retrieving contents in the web that fit some top goals, usually stated by users.

The chapter first presents a big picture of the framework, and a top-down overview of each level. Section 3.2 defines the discovery framework which is followed in the rest of the dissertation. Section 3.3 proposes the agent model that

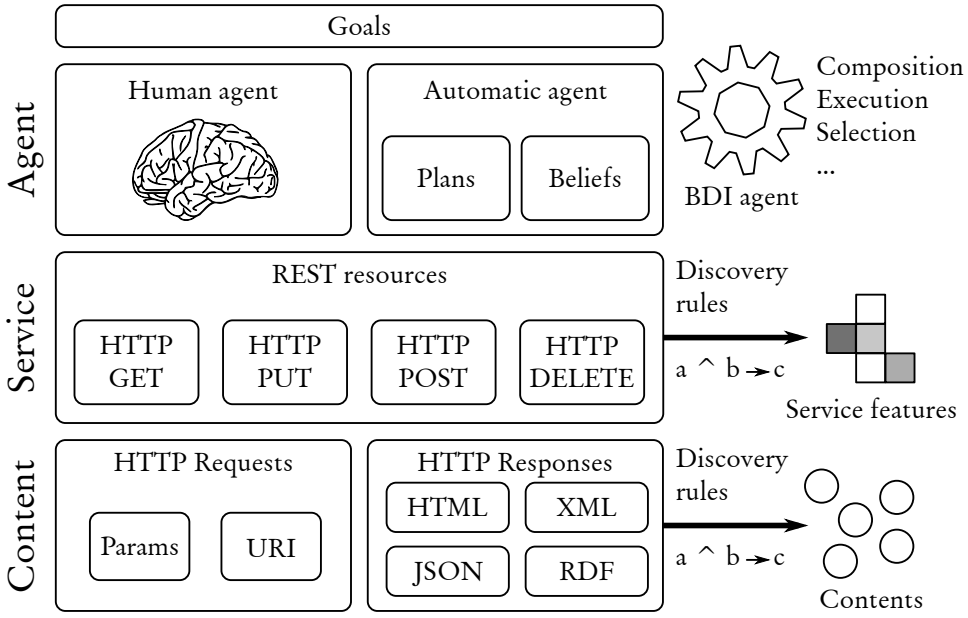

Figure 3.1: Discovery framework

builds plans for smart discovery of contents, while sections 3.4 and 3.5 describe the service- and content-level discovery, respectively.

3.2 Framework overview

The integrated framework for content and service discovery is defined in this section. We understand discovery as the process of identification and construction of an element's semantically meaningful description at some particular level. The framework is shown in figure 3.3. It is stacked on top of the REST architectural style [Fielding, 2000], the architectural style the one the World Wide Web is based on. The framework considers three levels of abstraction on the content and services that are available on the web. They are listed next from top to bottom:

- *Agent level*. This layer comprises the orchestration of services for fulfilling a user goal. Searching blogs to obtain relevant information about a product, look for the best price and suggest the user which are the best ones is an example of an orchestrated plan that is executed by an agent (either human or machine) attempting to reach a particular goal.

- *Service level*. This layer comprises the services that agents are able to use in

the Web, which nowadays vary from search services to booking services, social networking ones, etc. These services are generally orchestrated on the higher layer. They make use of contents in the lower layer by exchanging requests and responses with representations of resources that are present in the web.

- *Content level*. This level comprises the requests and responses that are exchanged between clients and servers when interacting with web resources. In the REST architectural style, requests consist mainly of a verb and a URI, optionally with parameters, while responses vary in format, although in the web it will usually be HTML.

Discovery will take place at these mentioned levels. At the service level, REST resources would be the discoverable elements to be analyzed, with semantic service descriptions being obtained. Discovery of data at the content level would produce meaningful semantically-annotated information.

Therefore, a method to extract semantic descriptions out of unstructured data at every level of the framework will be described. A uniform approach that employs first-order logic rules is employed to model discovery rules that allow to identify features on all levels. The combination of these features will comprise a semantic description for a discovered element.

In the next sections we will describe the agent architecture that operates at the orchestration layer and composes plans for fulfilling goals. Also, the techniques that allow to perform discovery for already existing pieces of contents and services will be described.

3.3 Agent model

The agent model that is stacked on top of the service level in the discovery framework is defined in this section. The agent model makes use of the REST architectural style through the semantically annotated services and contents. As shown on the general framework in figure 3.3, the agent is designed to perform the same tasks as a regular, human user of the web.

This problem statement leads to the agent being able to browse the web and run services just in the same way a human user would, with discovery rules as the means for extracting semantic descriptions out of regular resource representations. The agent would attempt to achieve a top level goal in the same way as human

users, e.g. finding a fact, a place or pictures about a particular person.

Additionally, the agent is able to manage the lower levels' discovery logic, i.e. manage service and content discovery rules. Discovery rules can be the result of a machine learning algorithm performed on supervised data. This supervised data can be added manually as training data set that is processed in a later stage. Further on, by introspecting into the discovery rules, the agent can anticipate the content and services that can be extracted out of resources, thus modifying its behaviour without interacting with those resources.

3.3.1 Architecture

The agent follows the Belief-Desire-Intention (BDI) pattern, which differentiates the independent modules that comprise a reactive system that interacts with other systems. In our case, beliefs are RDF contents that are extracted out of web pages. The agent has plans that represent the possible actions that it can perform, such as executing discovered services or visiting links, while the intentions are stacks of these plans to reach a particular goal. These top-level goals therefore represent discovery targets, i.e., contents that the agent attempts to add to its knowledge base.

Thus, both beliefs and goals are sets of RDF triples, while intentions are stacks of plans that are fired upon the creation or deletion of triples in the beliefs and goals triple sets. This makes up a naïve adaptation of AgentSpeak's agent model [Rao, 1996] to RDF, which results in an agent model that is similar to approaches that integrate BDI and Semantic Web standards with BDI agents [Laclavık et al., 2006]. The resulting architecture is shown in figure 3.2.

3.3.2 Plans

As long as the agent makes use of a RESTful architecture, it is able to interact with resources by exchanging requests and responses. This allows following hyperlinks and executing services such as web forms. Therefore, the four main HTTP methods make up the set of actions employed by the agent. Plans are defined around these actions to specify the possible behaviour of the agent to reach its goals. They consist of a triggering condition and a set of consequences, either subplans or actions.

The set of plans can be extended according to different domains in order to establish domain-specific behaviours, especially upon the presence of certain

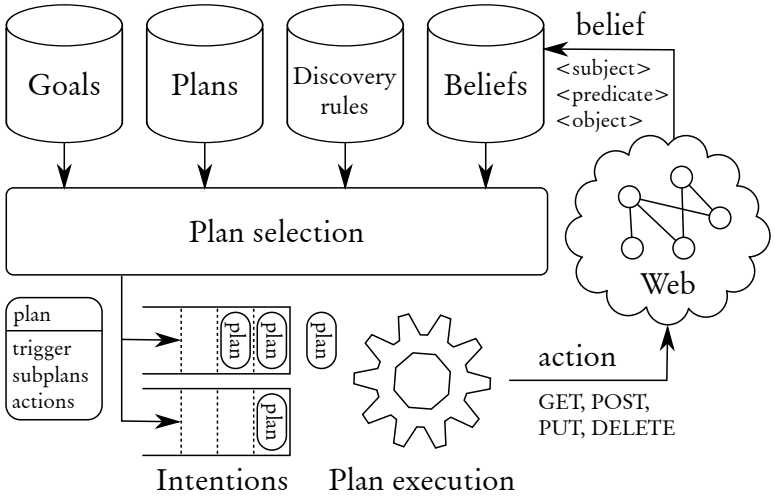

Figure 3.2: Agent model

services in the considered system. However, a base set of plans is defined here in order to provide the basic discovery capabilities of the agent.

Focused crawling plan

Whenever a resource's representation is expected to have contents with triples that are present in the goal set (trigger), perform a GET on that resource (action). This way, the agent will crawl the web in a greedy basis looking to fulfil its top-level goals. This can be done thanks to the content discovery rules in the knowledge base, which allow anticipating the contents to be found in a resource's representation. In AgentSpeak language, this plan is represented as[1]:

$$
\begin{aligned}
+![x, rdf{:}type, t] : \quad & content\_rule(y, x) \wedge \\
& [y, sc{:}type, t] \\
& \leftarrow get(x)
\end{aligned}
\tag{3.1}
$$

Note that, according to the Scraping Ontology [Fernández-Villamor et al., 2011], which will be introduced in section 3.5, $sc{:}type$ predicate indicates that, in a triple $< a, sc{:}type, b >$, the HTML fragment a is of type b after the extraction is performed.

[1]The syntax $[subject, predicate, object]$ has been used to represent RDF triples, which are not considered in the AgentSpeak language.

Deep web crawling plan

Whenever a keyword-filtered retrieval resource is expected to provide results that meet triples in the goal set (trigger), perform a GET on that resource. This allows using search forms to look for desired contents. The keywords that are entered in the form is the label of the resource, as a naïve conversion of RDF into natural language, which is subject to finer grain definition in domain-specific plans. In AgentSpeak, this plan is represented as:

$$
\begin{aligned}
+![x, rdf{:}type, y] : \; & [z, \; ms{:}has\_feature, \\
& \quad ms{:}Retrieval] \wedge \\
& [z, \; ms{:}has\_feature, \\
& \quad ms{:}KeywordFiltered] \wedge \\
& [z, \; ms{:}outputs, y] \wedge \\
& [x, \; rdfs{:}label, l] \\
& \leftarrow \; get(z, \{(keywords, l)\})
\end{aligned}
\tag{3.2}
$$

3.4 Service level

The discovery at service level involves building semantic service descriptions out of the HTTP interaction data and the discovered semantic contents from the lower level in the proposed discovery framework. The input service model is therefore comprisen by the output content model and the HTTP interaction data, i.e. the involved URI, the HTTP method and the HTTP parameters.

A lightweight service description model that represents the semantics of discovered services [Fernández-Villamor et al., 2010b] is employed for this task, and is described in the next section. This service model is used as the output model for this layer, i.e. the model that discovered elements (in this case, services) use.

3.4.1 Description model

The output service model used for service discovery applies ideas inspired in mix-ins, Aspect-Oriented Programming (AOP), and Feature-Oriented Programming (FOP) paradigms to semantic service description, as will be detailed in chapter 5. These paradigms extend Object-Oriented Programming (OOP) by allowing the modelling of secondary concerns in an isolated way. Services are modelled in a similar way in the service level of the framework to facilitate their discovery out of semantically discovered contents.

A service is modelled as a set of features f_1, f_2, ..., f_i that it has. In web applications, some examples of service features are "*performing a retrieval operation*", "*requiring user authentication*", "*performing a storage operation*", "*handling images*", or "*outputting a set of resources*". This allows reusing feature descriptions in services that are different but have one or more features in common. The library of features comprise the output service model.

The output service model is therefore a vocabulary of terms that can be extended, each term representing a feature. As the framework follows the REST architectural style, terms for HTTP GET, POST, PUT or DELETE requests are part of the vocabulary.

An example of service description is `keyword-filtered multiple picture get`, which describes a search service of pictures that are filtered by keywords. Four terms represent the features that are used in order to define the service: `keyword-filtered`, `multiple`, `picture`, and `get`. Because of following the REST architectural style, at least a term that represents the underlying HTTP method used has to be included in the description.

An extended service description can be built by combining term definitions. Definitions can be tied to more than one term. For example, there can be a definition for `get`, a definition for `picture`, but also a definition for `picture` and `get` altogether. Some possible definitions for terms `keyword-filtered`, `picture`, `multiple`, and `get` are shown next:

$$\begin{aligned} & method(x) = GET \wedge \\ & status(x) = 200 \Rightarrow \\ & ms{:}has\_feature(x, get) \end{aligned} \tag{3.3}$$

$$\begin{aligned} & |Output(x)| > 1 \wedge \\ & ms{:}has\_feature(x, get) \Rightarrow \\ & ms{:}has\_feature(x, multiple) \end{aligned} \tag{3.4}$$

$$\begin{aligned} \forall k, y(\quad & k \in Input(x) \wedge \\ & y \in Output(x) \wedge \\ & dc{:}subject(y, k) \wedge \\ & ms{:}has\_feature(x, get)) \Rightarrow \\ & ms{:}has\_feature(x, keyword\_filtered) \end{aligned} \tag{3.5}$$

$$\begin{aligned}\forall y(\quad & y \in Output(x) \wedge \\ & rdf{:}type(y, foaf{:}Image) \wedge \\ & ms{:}has\_feature(x, get)) \Rightarrow \\ & ms{:}has\_feature(x, picture)\end{aligned} \tag{3.6}$$

By combining the definitions for a service x with the previously mentioned description `keyword-filtered multiple picture get`, an extended description would be the following:

$$\begin{aligned}\forall k, y(\quad & method(x) = GET \wedge \\ & status(x) = 200 \wedge \\ & |Output(x)| > 1 \wedge \\ & k \in Input(x) \wedge \\ & y \in Output(x) \wedge \\ & dc{:}subject(y, k) \wedge \\ & y \in Output(x) \wedge \\ & rdf{:}type(y, foaf{:}Image))\end{aligned} \tag{3.7}$$

The result of wrapping services with the provided semantic feature-based descriptions is:

- Searchability. Service descriptions can be advertised as Linked Data by publishing their RDF description, which allows being processed by an automatic agent.

- Testability. Preconditions and postconditions for each feature can be checked when executing a service, which allows checking the service's correct execution.

- Execution. Thanks to the semantic description, the services have a known interface which can be used by automatic agents to run the services.

As a feature-oriented description framework, this approach has the following advantages: (i) it allows the reuse of feature descriptions among different services, and (ii) it reduces the description task to selecting a set of features that describes the considered service, given a vocabulary of terms.

3.4.2 Service discovery rules

In order to perform discovery, there must exist a mapping between the service's feature set and the service input model. To achieve this, feature definitions are used as service discovery rules. Thus, discovery rules map a set of features f_1, ..., f_k to a set of conditions, defined using the output content model and the HTTP interaction data. For example, given the features f_1 ("outputting a set of resources") and f_2 ("handling images"), some discovery rules would just be the feature definitions shown on equations 3.4 and 3.6.

Discovery rule on equation 3.4 formalizes the feature of term $multiple$ by stating that the service's output cardinality has to be higher than one. Meanwhile, discovery rule on equation 3.6 formalizes that all output resources are images, and is applicable for services that (i) output a set of resources (as of feature f_1) and (ii) handle images (feature f_2). Allowing definitions that are activated upon the presence of more than one feature might be regarded as unnecessary complexity. However, this serves to resolve the issue of feature interaction, already identified in feature-oriented programming [Prehofer, 1997].

With these mentioned discovery rule, a service's features can be identified after knowing the semantics of the contents involved in the HTTP interaction. Further details on the service model and the automatic construction of service discovery rules will be given in chapter 5.

3.5 Content level

A growing amount of data is available to users in the web. Web users can enjoy plenty of services and information in e-commerce web sites, electronic newspapers, blogs and social networks. Although this data is available for its consumption by users, its format is not suited for automated agents and computer programs. This has favoured the research in several fields such as web content mining [Kosala and Blockeel, 2000] or Semantic Web [Berners-Lee et al., 2001], that seek manners to build linked interoperable data that can be automatically processed by software systems.

Several approaches such as Linked Data initiative [Bizer et al., 2009] are favouring the publication of annotated data in web resources, so that automatic processes can actually consume this data and perform other operations. Similarly, other research fields attempt to take advantage of this amount of available information, such as mashup applications. However, ontologies and applications that expose

their data are not widespread, constraining the Linked Data initiative, mashups and service composition.

The field of Web Content Mining applies data mining techniques to the discovery and extraction of information available on the Web. Web Content Mining comprises several research fields such as Information Extraction or Natural Language Processing, which research related techniques that are used to extract data from web documents [Chang et al., 2006].

Approaches to the problem of extracting information out of HTML documents considers processing either the DOM tree or the resulting rendering information. The first approach involves defining an extractor or wrapper [Kushmerick, 1997] [Kushmerick, 2000] that selects the relevant information out of the DOM tree. The latter is a vision-based approach that attempts to provide a more general solution to the problem by assuming that similar content types have similar visual features [Wei et al., 2006] [Cai et al., 2003].

The lowest level of the framework is the content level, where content is discovered out of unstructured web resources. The next section describes content level discovery rules.

3.5.1 Semantic Scraping approach

As said, the web is a hypermedia system that follows the REST architectural style [Fielding, 2000]. When a client accesses a web resource on a server, the server returns a representation of the resource. Usually, these representations are formatted in HTML, a language that allows defining the structure of a document for its rendering on a web browser. HTML documents are structured as a DOM tree, which defines the logical structure of the HTML document that will be used for rendering the representation on a web browser. In order to have information about the resource's content and not about its rendering structure, Linked Data proposes using resources' representations that include metadata, by enhancing HTML with semantic annotations or by providing RDF representations.

Whenever a resource provides unannotated HTML, a technique that processes the DOM tree in some way needs to be used to identify the structure of the data present in the HTML document and build the associated RDF graph. Also, in a web resource there are DOM fragments that do not provide information, such as advertisements, headers, footers, or decorative elements, while other fragments such as posts or comments have valuable information. In our framework, discovery rules will be employed to identify what pieces of information are relevant in

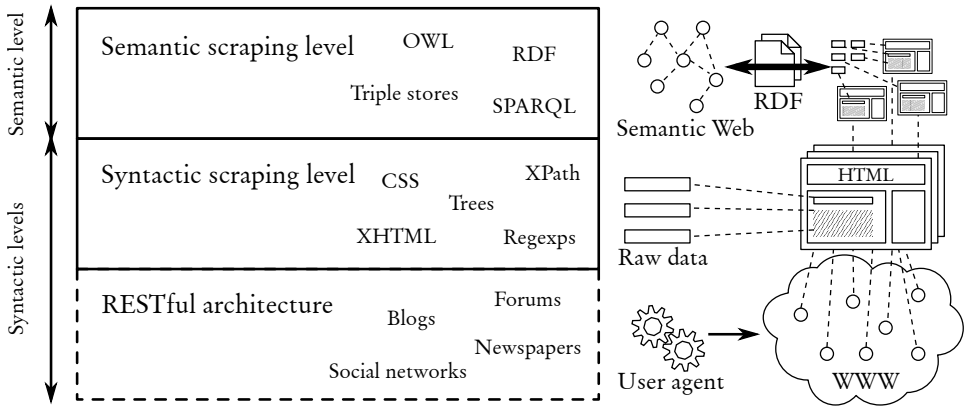

Figure 3.3: Semantic scraping approach

a web resource and to identify what relations are stated in a web fragment. For instance, a heading in a piece of news might represent the news title. A discovery rule will use Content Style Sheets (CSS) information, rendering information or NLP to identify the relevant data in the resource's representation.

Therefore, the input model that discovery rules use at this level comprise HTML fragments, which identify relevant pieces of data in a document, and selectors, which are any mean to identify a fragment inside a document. Usually, web scrapers use regular expressions or CSS or XPath selectors to achieve these tasks, while the output of a web browser when rendering a web fragment, which consists of a set of properties such as typeface, color or dimensions, can also be used through visual selectors.

On the contrary, the output model is comprisen by the different types of contents that are available in the web. Ontologies like Semantically-Interlinked Online Communities Project (SIOC), Friend of a Friend (FOAF) or Dublin Core (DC) address this issue by defining schemas for the modeling of blog posts, relationships between users or annotation of metadata in publications, thus comprising the output content model of our discovery framework.

3.5.2 Semantic scrapingontology

An ontology for semantic scraping is proposed in this section. The approach to using semantics for contents extracted from the web is shown in figure 3.3. The model considers three levels of abstraction in order to provide an integrated model for semantic scraping:

- Semantic scraping level. This level defines a model that maps HTML fragments to semantic web resources. By using this model to define the mapping of a set of web resources, the data from the web is made available as knowledge base to scraping services. This level provides semantics to the syntactic scraping capabilities of the level below.

- Syntactic scraping level. This level gives support to the interpretation to the semantic scraping model. Wrapping and Extraction techniques such as DOM selectors are defined at this level for their use by the semantic scraping level.

The model is stacked on top of the REST architectural style. The additional semantics and data mappings that are necessary to allow information scraping on a RESTful architecture are defined by the upper levels of our framework.

On top of the semantic scraping level there could exist scraping services that make use of semantic data extracted from unannotated web resources. Possible services that benefit from using this kind of data can be opinion miners, recommenders, mashups that index and filter pieces of news, etc. In the case of the framework being proposed in this chapter, there would be services that make use of the semantically annotated contents.

The paradigm behind scraping services has subtle differences from that behind traditional Semantic Web applications or knowledge-based systems. While annotated data in the Semantic Web allows automatic knowledge extraction and retrieval by automatic agents, data in unstructured web documents require prior supervision of some kind to allow information extraction. This implies that when designing a scraping service, the following steps are required:

- Scraping data identification. Data that wants to be scraped and merged with other knowledge is identified in this task. Target web sites and resources are identified for fragment extraction.

- Data modelling. A model to represent the extracted data is defined in this task. Either existing ontologies might be available or new ones should be defined. The result from this step is an ontology that fits the data that needs to be extracted. A bounded context, i.e. a conceptual context where a domain model has a non-ambiguous meaning, should be identified in order to separate domain models of similar fields. Methodologies for the definition of ontologies can be useful for this task.

- Extractor generalization. In order to perform massive extractions, enough samples need to be collected to generalize an appropriate extractor. This collection of samples needs to be provided to a human administrator or an automated or semi-automated module. Using this data set, one or more extractors are defined at the semantic scraping level and serve to provide additional knowledge to the scraping service.

Let's consider a movie recommender that requires extracting data from the Internet Movie Database[2]. Data about reviews are added to the recommender's knowledge in order to enable collaborative filtering of movies. Reviews and user reviewers are therefore the identified data to scrape. As long as an existing movie ontology is defined, no ontology modelling would be needed. Also, in case extractors are built automatically using a machine learning approach, data samples should belong to the bounded context of cinema and movies.

Semantic scraping

Semantic scraping defines the mapping between web data and semantic web resources. An RDF model that allows formalizing this mapping has been defined, and is called the Scraping Ontology[3].

Applying the model to the definition of extractors of web resources allows separating the declarative from the procedural model in the web content extraction process. This enables implementing technology-independent extractors or automating certain tasks such as focused and personalized scraping.

The Scraping Ontology allows to reference HTML fragments in RDF and define web content extractors, being a basis for the programmatic definition of extractors for screen scraping. This requires bridging the gap between both RDF and HTML's data models. HTML is a markup language for documents with a tree-structured data model. On the other hand, RDF's data model is a collection of node triples, defined by a subject, a predicate, and an object. Each node can be a text literal, a resource (identified by a URI) or a blank node.

A model comprisen of a vocabulary of RDF terms has been defined to represent HTML fragments and their mapping to RDF resources. This serves as a model for the discovery framework's content level. A summary of the model is shown in figure 3.4. The basic classes of the model are described next:

[2] `http://imdb.com`
[3] `http://lab.gsi.dit.upm.es/scraping.rdf`

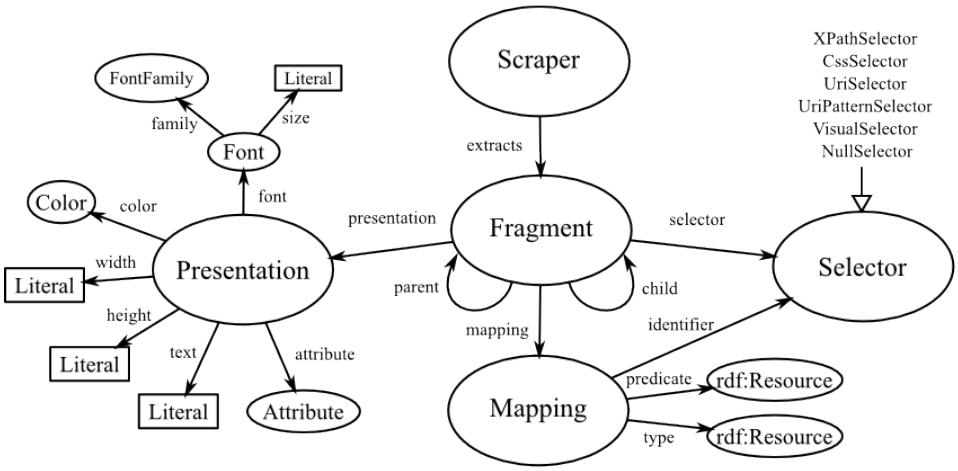

Figure 3.4: Semantic scraping RDF model

Scraper A scraper is an automatic agent that is able to extract particular fragments out of the web.

Fragment Any element of an HTML document. It serves to represent and traverse a whole subtree of a document.

Selector A condition that indicates which this element is. Different selector terms are defined for each selector type. Selectors can be XML Path Language (XPath) expressions, CSS selectors, URI selectors, etc. Selectors are means to identify a web document fragment.

Mapping The mapping between a fragment and an RDF resource or blank node. An identifier is defined to map the fragment to a URI. A predicate between the parent's mapped fragment and this is defined to produce an RDF triple. Also, an RDF class can be assigned to the mapped resource of this fragment.

Presentation The representation of a fragment. This includes HTML attributes as well as visual parameters such as color, size or font.

The proposed vocabulary serves as link between HTML document's data and RDF data by defining a model for scraping agents. With this RDF model, it is possible to build an RDF graph of HTML nodes given an HTML document, and provides semantics to syntactic scraping.

The objective of this RDF model should not be confused with that of RDFa. RDFa defines a format for marking up HTML elements to extract an RDF graph. Our model complements RDFa by allowing RDF graphs to refer to data that is present in HTML fragments in an unannotated HTML document.

Syntactic scraping

Syntactic scraping comprises the required technologies to extract data from web resources. Some of the considered scraping techniques are the following:

- CSS selectors. CSSs define the visual properties of HTML elements. These visual properties are mapped to elements through the use of CSS selectors, defined through a specific language. Therefore, CSS is one technology that serves to select and extract data.
- XPath selectors. Similarly to CSS selectors, XPath[4] is a different language for HTML node selection.
- URI patterns. URI patterns allow to select web resources according to a regular expression that is applied on the resource's URI. While XPath or CSS selectors are able to select an element at document level, URI patterns allow selecting documents, i.e. resources representations, according to the resource's URI.
- Visual selectors. Visual information can be used to select nodes. HTML nodes are rendered with a set of visual properties given by the used browser. It is common that human users prefer uniform web designs. Web designers thus make elements of a same kind to be rendered with similar visual properties to help identification. A visual selector is a condition that combines several visual properties of an element to identify the element's class.

Other kinds of selectors that process HTML's inner text are available as well and fit into the model. This way, extractions from natural language parsing or text tokenization are possible.

Selectors at the syntactic scraping level allow to identify HTML nodes. Either a generic element or an unambiguously identified element can be selected using these techniques. Their semantics are defined in the upper semantic scraping level, allowing to map data in HTML fragments to RDF resources.

[4] `http://www.w3.org/TR/xpath/`

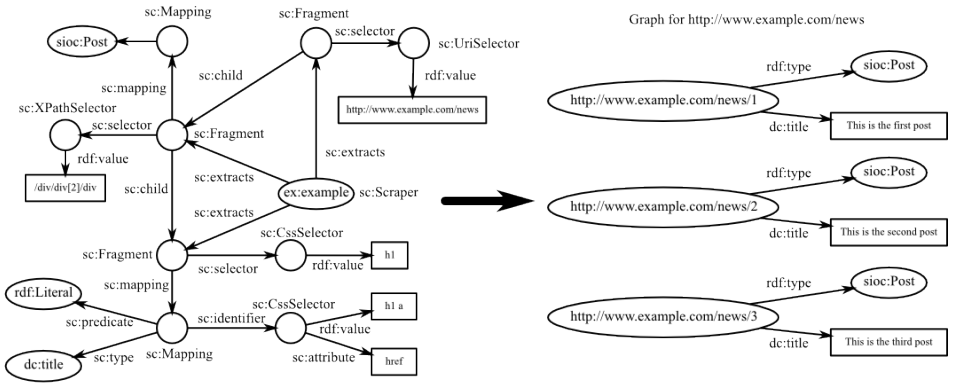

Figure 3.5: Example of semantic scraper

An example of the usage of selectors for a news scraper is shown in figure 3.5. In this case, a scraper is defined that is able to scrape a set of posts (by using the SIOC ontology [Breslin et al., 2006]) from a specific URI. A sample mapped RDF graph is shown in the figure, too.

3.5.3 Content discovery rules

This section introduces content discovery rules, once the Scraping Ontology has been presented. At the content level, discovery rules allow to identify data in web resources and map them to the RDF. For this, they use as input model the Scraping Ontology, thus using selectors to identify the different fragments of data. The output model is comprised by the different ontologies available in the Linked Data cloud.

An example of content discovery rule is given next:

$$\begin{aligned} \forall x,y(\quad & uri(x,http://nytimes.com) \wedge \\ & parent(x,y) \wedge \\ & css(y,".story\ h2\ a")) \Rightarrow \\ & rdf{:}type(sioc{:}Post,y)\ ) \end{aligned} \tag{3.8}$$

This sample rule defines that all DOM tree nodes that satisfy a particular CSS selector in New York Times home page are posts, according to SIOC ontology [Breslin et al., 2006].

The main challenge at the content level is defining discovery rules which are

robust and generalizable. A *robust* rule is one that extracts the same data even with changes in the DOM tree of the web resource. If a rule is not robust, it might stop working once the layout of a web site is changed by its web administrator on a redesign stage [Lerman et al., 2003]. A rule that *generalizes* is one that is valid for all the web resources that contain the same kind of data. If a rule is only valid for the web resource (or resources) that it was defined for, it does not generalize across different resources. The main limitation of wrapper induction is that wrappers are only valid for the web pages they were designed for [Kushmerick, 1997]. Using NLP and visual selectors as input content model improves generalization capabilities of the discovery rules [Cai et al., 2003]. This problem will be covered in more detail in chapter 4.

3.6 Conclusions

Throughout this chapter a framework for the discovery of services and contents in the web has been proposed. An agent model that fits the discovery framework is also defined for the implementation of combined services or contents in cases where discovery is required. With this agent model, an agent is able to perform discovery tasks whenever appropriate and plans thanks to the semantic descriptions of the discovered elements and the agent's ultimate goals.

A feature-based service description approach has been introduced as an approach to simplify service description. These kinds of service descriptions are simple and allow automating various tasks in order to push service automation in the semantic web, thus offering uniform interfaces, discoverability, and automating validation.

Additionally, an RDF model for web scraping has been defined at the content discovery level. This enables an open framework for web scraping. The tasks of building an RDF graph out of a web document have been shown. With this, a semantic screen scraper has been developed. The semantic screen scraper produces RDF graphs out of web documents and RDF-defined extractors, that offer interoperable scraping information.

In chapters 5 and 4, the automatic construction of service and content discovery rules, respectively, will be covered.

Chapter 4

Content discovery

This chapter describes the automatic induction of content discovery rules. Information extraction out of web pages, commonly known as screen scraping, is usually performed through wrapper induction, a technique that is based on the internal structure of HTML documents. As such, the main limitation of these kinds of techniques is that a generated wrapper is only useful for the web page it was designed for. To overcome this, in this chapter it is described an algorithm that generates content discovery rules that can be used to extract data from web pages. These rules are based on visual features such as font size, elements positioning or types of contents. Thus, they do not depend on a document's internal structure, and are able to work on different sites.

4.1 Introduction

The vast amount of information available on the Web turns it into an important knowledge source for many different domains. Semantic Web standards [Berners-Lee et al., 2001] and the Linked Data initiative [Bizer et al., 2009] propose the annotation of web resources with metadata, which allows the processing of web resources by automated agents. Despite the growth in adoption of standards of this kind, many web sites still do not provide means to retrieve their contents according to a known, structured schema. For example, out of 17 popular electronic newspapers surveyed[1], none of them provide semantic annotations of a Semantic Web standard.

Examples of applications that make use of web data can be travelling mashups, which scan web pages for flights, hotels and trains, and provide the best trip plan according to a user's preferences. Those flight, hotel and train web sites that adopted the Linked Data initiative would publish metadata that allows a simple extraction of these sites' data. However, in order to get data from other sites that do not publish appropriate metadata, it would be neccesary to use Screen Scraping techniques to get access to data that is published in an unstructured way [Chang et al., 2006] [Kosala and Blockeel, 2000].

Traditional scraping approaches are based on some kind of DOM[2] tree processing. Usually, techniques such as tree-to-tree edit distance [Bille, 2005] [Barnard et al., 1995] [Chen, 2001] and wrapper induction [Kushmerick, 1997] [Kushmerick, 2000] are used to, either manually [Hogue, 2005] or automatically [Crescenzi et al., 2001], build wrappers that allow extracting data from web resources. The main limitation of DOM tree processing is that these wrappers are specific to one web site, and therefore do not show generalization capabilities for extracting data from other visually similar web sites. Wrappers also require being rebuilt, as part of a maintenance process, when a web resource layout changes [Lerman et al., 2003]. Alternatively, other approaches consider processing visual properties of DOM elements when rendered by a web browser [Wei et al., 2006] [Pembe and Güngör, 2010]. The advantage of these kinds of approaches is its generalization across different sites.

This chapter describes a system that performs extraction of Linked Data out

[1] The surveyed newspapers were New York Times, Wall Street Journal, The Guardian, The Telegraph, Spiegel, Bild, Frankfurter Allgemeine Zeitung, Le Monde, L'Équipe, ABC, El Mundo, El País, ADN, 20 Minutos, Público, Marca, and As.

[2] `http://www.w3.org/TR/DOM-Level-2-Core/`

of web resources and which shows high generalization capabilities and robustness. Semantic information in a web resource is a graph that, following Semantic Web's standards, can be represented using the RDF [Lassila and Swick, 1999]. Therefore, extracting RDF data implies building the associated graph out of the information present in the web page. We propose using first-order logic rules to extract RDF graphs. To build these rules, we have built an algorithm which follows a specific-to-general basis. First, the information to be extracted is manually identified in web pages and with these samples a set of overfitting rules are built. Then, the algorithm combines and generalizes rules progressively. This supervised first-order logic classifier makes use of web elements' visual properties. Therefore, the knowledge acquired by the classifier generalizes across web sites and is robust to layout changes on them.

4.2 Problem statement

The Web is a hypermedia system that follows the REST architectural style [Fielding, 2000]. When a client accesses a web resource on a server, the server returns a representation of the resource. Usually, these representations are formatted in HTML, a language that allows defining the structure of a document for its rendering on a web browser. HTML documents are structured as a DOM tree, which defines the logical structure of the HTML document (see Fig. 4.1) that will be used for rendering the representation on a web browser. In order to have information about the resource's content and not about its rendering structure, Linked Data proposes using resources' representations that include metadata, by enhancing HTML with semantic annotations or by providing RDF representations, such as in Fig. 4.1. Such figure shows the RDF representation of a piece of news by using SIOC ontology [Breslin et al., 2006] and DC schema [Weibel, 1997], ontologies chosen due to their high adoption and popularity among the Semantic Web community.

Whenever a resource provides unannotated HTML, a technique that processes the DOM tree in some way needs to be used to identify the structure of the data present in the HTML document and build the associated RDF graph. This process is known as Screen Scraping, and it implies solving the following problems:

- Identifying what pieces of information are relevant in a web resource. Usually, in a web resource there are DOM fragments that do not provide information, such as advertisements, headers, footers, or decorative elements,

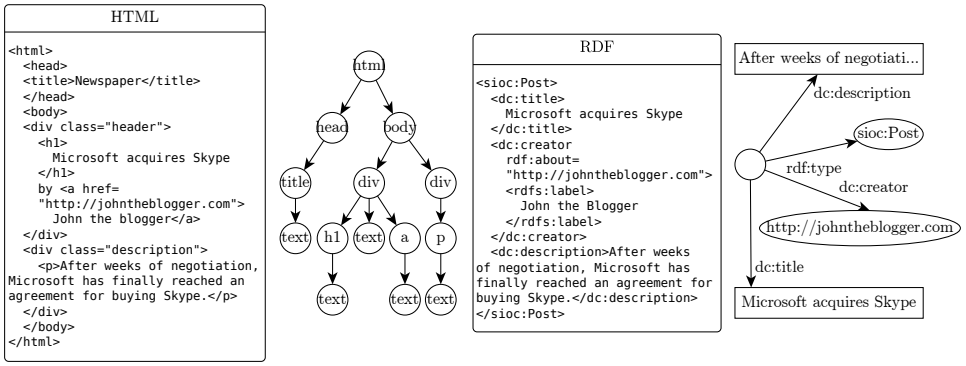

Figure 4.1: HTML vs RDF documents

while other fragments such as posts or comments have valuable information.

- Identifying what relations are stated in a web fragment. For instance, a heading in a piece of news might represent the news title.

The conceptual model behind the process of building an RDF graph out of an HTML page is shown in Fig. 4.2 and serves as a basis for addressing the problem of web resource screen scraping. It shows the elements involved in the process of Screen Scraping in order to familiarize the reader with the process. The figure shows the relations among these elements and how a scraper requires different pieces of information to complete the process of converting a web page into a set of RDF resources. As shown, the main elements involved in the problem of Screen Scraping are:

Web page The HTML representation that is returned by a web browser when attempting to retrieve a web resource. Web pages are designed to be used by human users through a web browser.

Fragment Any web fragment inside a web page, or a web page itself. A web page fragment usually shows information about one or more concepts, such as a blog post, a flight, a web result, etc.

Selector Any mean to identify a fragment inside a document. Usually, web scrapers use regular expressions or CSS or XPath selectors to achieve these tasks.

DOM element Each of the elements in the DOM tree of an HTML document.

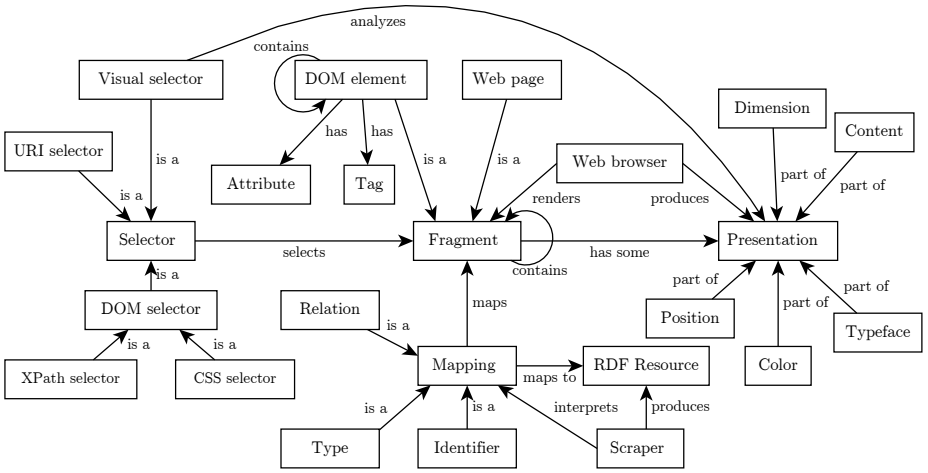

Figure 4.2: Scraping conceptual model

They represent the hierarchical structure of the document, and can be referenced through the usage of DOM selectors.

Presentation The output of a web browser when rendering a web fragment, which consists of a set of properties such as typeface, color or dimensions. This output is used by users to interpret the contents of a web page, and can be used by visual selectors as well to identify web fragments according to their visual properties.

Mapping The mapping that exists between a fragment inside an HTML document and the RDF resource it represents. A mapping might consist of stating a predicate about a resource or that a resource has a particular URI.

Scraper An automated process that is able to interpret mappings to produce RDF data. An RDF document that defines the extraction mappings on web fragments can be used by the appropriate scraper to extract the data from a web resource in RDF format.

Several challenges are involved behind the problem of Screen Scraping. The main difficulties are listed next:

- Identification of data to extract. First, the desired data to be extracted needs to be defined. Either tools for manual annotation or automated approaches

that compare similar pages or analyse documents' structure are used for this task.

- Definition of selectors. After the data to extract have been identified, appropriate selectors need to be constructed. Either regular expressions, wrappers, CSS, XPath, or visual selectors can be used. The type and quality of the defined selector will affect its applicability to other sites.

- Changes in web pages. Whenever a web page's layout changes, the defined selectors can be not valid anymore. The consequences are usually badly extracted data or no extracted data at all. Quality checks and better selectors help to prevent this from happening.

- Dynamic, JavaScript-intensive web pages. Some web sites change the layout after page load or after interaction with the user. A solution consists of executing JavaScript and reproducing the interactions with mocked-up users to access those data, although overcoming this problem is out of the scope of this thesis.

Scraping mappings contain the selectors that identify data and their generalization capabalities. Therefore, when employing mappings to tackle the problem of Screen Scraping, the main problem is defining quality mappings that allow a scraper to extract the data from a web resource. Then, we identify two desireable aspects on the definition of a mapping, which are obtaining *robustness* and *generalization*:

- A *robust* mapping is one that extracts the same data even with changes in the DOM tree of the web resource. If a mapping is not robust, it might stop working once the layout of a web site is changed by its web administrator on a redesign stage [Lerman et al., 2003].

- A mapping that *generalizes* is one that is valid for all the web resources that contain the same kind of data. If a mapping is only valid for the web resource (or resources) that it was defined for, it does not generalize across different resources. The main limitation of wrapper induction is that wrappers are only valid for the web pages they were designed for [Kushmerick, 1997].

Therefore, the problem that is addressed in this chapter is building robust and generalizable extraction mappings that allow scraping web resources. Metrics to measure robustness and generalization will be given in section 6.2.2.

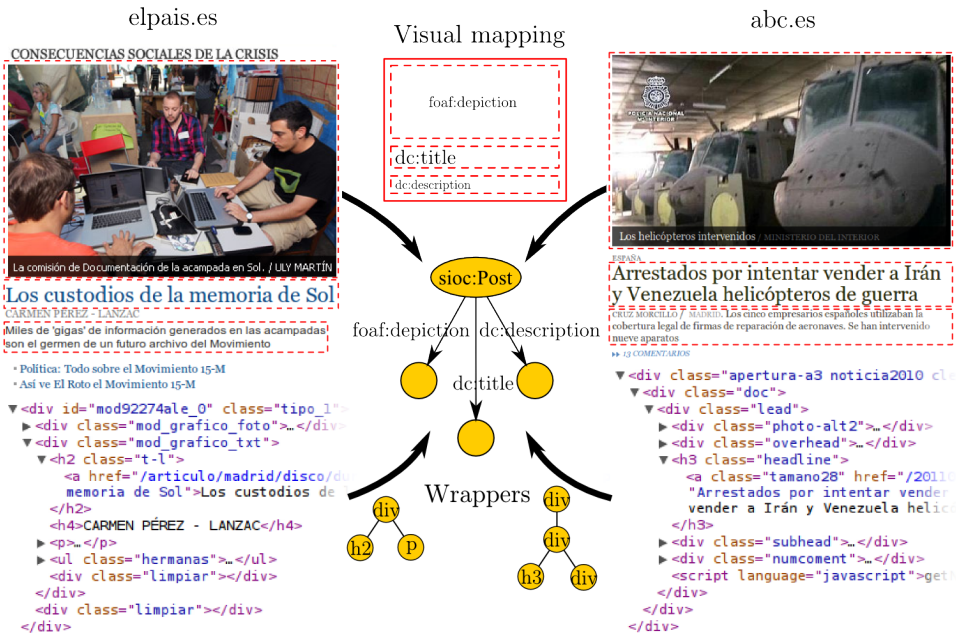

Figure 4.3: Conversion of a DOM tree into an RDF graph

4.3 Rule induction for content extraction

In this section, we describe an approach to automatically create mappings for extracting RDF data from HTML documents while attempting to solve the issues described above. The approach uses visual features of the elements displayed in a web browser in a way as shown in Fig. 4.3, which compares the approach of using wrappers against visual selectors. The figure shows two Spanish news sites (Abc and El País), which have different layouts but similar looks. When using the techniques behind wrapper induction, it is required to define a new wrapper for each different web site, as they are based on the DOM tree structure of a web site, which is rarely shared among different web sites. Using techniques that are based on visual features allows mappings to generalize accross different web resources, as these web resources share similar looks.

As introduced in section 3.5.2, the Scraping Ontology is an RDF schema that allows defining mappings between HTML elements and the RDF data the mappings represent [Fernández-Villamor et al., 2011], and is used in this chapter to represent the RDF mappings. This ontology contains the terms that were defined in the conceptual model.

The mappings defined in this ontology are sequences of fragments with the RDF data that they represent. An example of the mappings that are considered by the algorithm is shown next:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:sc="http://lab.gsi.dit.upm.es/scraping.rdf#"
   xmlns:sioc="http://rdfs.org/sioc/ns#">
  <sc:Fragment>
    <sc:type rdf:resource="http://rdfs.org/sioc/ns#Post"/>
    <sc:selector>
      <sc:VisualSelector>
        <sc:max_height>139</sc:max_height>
        <sc:max_relative_x>508</sc:max_relative_x>
        <sc:max_relative_y>1084</sc:max_relative_y>
        ...
      </sc:VisualSelector>
    </sc:selector>
    <sc:subfragment>
      <sc:Fragment>
        <sc:max_cardinality>1</sc:max_cardinality>
        <sc:min_cardinality>1</sc:min_cardinality>
        <sc:type
           rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal"/>
        <sc:relation
           rdf:resource="http://purl.org/dc/elements/1.1/title"/>
        <sc:selector>
          <sc:VisualSelector>
            <sc:font_family>serif</sc:font_family>
            <sc:max_font_size>24</sc:max_font_size>
            ...
          </sc:VisualSelector>
        </sc:selector>
      </sc:Fragment>
    </sc:subfragment>
    ...
  </sc:Fragment>
</rdf:RDF>
```

As it can be seen, not only is the output of extraction mappings expressed in RDF, but also the mappings themselves, as a result of using the Scraping Ontology. In the previous RDF document, a fragment that represents a news *post* (according to SIOC ontology) is defined. The fragment is selected out of a web resource thanks to a visual selector, for which some visual conditions are defined. Additionally, this news post has other subfragment, which is also selected thanks to another visual selector. This subfragment is related to the parent through a *title*

relation (according to DC schema).

These mappings can be represented as rules that are applied to web resources. If the rule succeeds, data is extracted from the web page. Two examples of rules are the following ones:

$$width(x) > 200 \wedge width(x) < 300 \wedge font\_size(x) < 14 \Rightarrow rdf{:}type(x, sioc{:}Post) \tag{4.1}$$

$$width(x) > 200 \wedge width(x) < 300 \wedge font\_size(x) < 14 \wedge parent(x,y) \wedge font\_size(y) > 16 \wedge font\_weight(y) > 400 \Rightarrow rdf{:}type(x, sioc{:}Post) \wedge dc{:}title(x,y) \tag{4.2}$$

Equation 4.1 shows a rule which states that an HTML fragment x represents a blog post if some font and size conditions are evaluated as true. The rule shown in equation 4.2 is a more complex one, and makes a statement about a post by considering it also has a subfragment that represents a title. If the conditions are evaluated as true, then RDF triples (i.e. data structures that consist of a subject, a predicate and an object) are built and thus a post with a title is generated as output. The rules make use of the already mentioned SIOC and DC ontologies to model the extracted data.

As seen, rules have a left-hand side with conditions about the visual features of HTML fragments (e.g., $width(x) > 200$ or $font\_size(x) < 14$), as well as structural conditions about how these HTML fragments are organized in the DOM tree (e.g., $parent(x,y)$). The right-hand side contains RDF statements about the HTML fragments involved in the left-hand side. More formally, the rules have the following structure:

$$\bigwedge_i c_i(x_i \in \mathscr{X}) \wedge \bigwedge_{i,j} parent(x_i \in \mathscr{X}, x_j \in \mathscr{X}) \Rightarrow \bigwedge_i T_i(x_1, ..., x_N) \tag{4.3}$$

where:

- $\mathscr{X} = \{x_1, x_2, ..., x_N\}$ is the set of HTML fragments involved in the rule.
- $c_i(x_j)$ is a condition on attributes of the HTML fragment x_j (e.g. $width(x_3) < 300$).
- $parent(x_i, x_j)$ is a binary predicate that states that x_i has to be a parent of x_j in the DOM tree (i.e. HTML fragment x_j is contained in fragment x_i).

- $T_i(x_1, ..., x_N)$ is an statement about the RDF resource represented by $x_i \in \mathcal{X}$ which can involve one or more of the variables x_i that are used in the left-hand side (e.g. $rdf{:}type(x_2, sioc{:}Post)$ or $dc{:}title(x_2, x_3)$).

4.3.1 Training attributes and classes

As said before, robustness and generalization are aspects that are desireable to have in extraction mappings. An RDF mapping can be defined using different selectors. Selectors such as CSS or XPath might result in extraction mappings that can only be applied on a reduced set of web resources. The usage of visual selectors allows extraction mappings to work on different web sites. The algorithm makes use of several visual features of the DOM tree elements present in a web resource, as listed next:

- Continuous attributes:
  - Positioning (X and Y).
  - Width and height.
  - Font size and weight.
- Discrete attributes:
  - Font family: sans, sans-serif or monospace.
  - HTML tag: link, image or other.

The values of these attributes are captured by using a web browser, which renders web pages according to CSSs and other files, such as images.

The classes of the samples can be any RDF triple. As have been shown previously, the rule examples used RDF properties such as $rdf{:}type$, $sioc{:}Post$ or $dc{:}title$, but any other kind of RDF triple could be employed.

4.3.2 Induction algorithm

The algorithm builds a rule set in a specific to general basis, by using overfitting rules that are combined into more general ones. This decision is taken in order to reduce the search space; in top-down induction of logical rules [Blockeel and De Raedt, 1998] all the possible combination of conditions need to be explored, while in rule combination approaches, such as ours, the search space is reduced to the possible combinations among similar rules [Domingos, 1995].

The algorithm requires a supervised dataset as input. There are many techniques that could be used to obtain such supervised database by performing an extraction from a set of web resources. It can be done by using a manually defined wrapper, so the typical techniques for wrapper induction can be used for this purpose. Once this is done, a training dataset is obtained and used as input for the induction algorithm.

Then, overfitting rules are built out of the results of the supervised extraction. An example of the overfitting rules is given with after the following training sample, classified as a $sioc{:}Post$:

$$\begin{aligned} width(x) &= 100 \\ height(x) &= 200 \\ font\_size(x) &= 12 \\ font\_type(x) &= sans \\ rdf{:}type(x &, sioc{:}Post) \end{aligned} \tag{4.4}$$

As it can bee observed, the sample represents an HTML fragment x which has some visual properties about size and font. This HTML fragment x is converted into the following overfitting rule:

$$\begin{gathered} width(x) \geq 100 \wedge width(x) \leq 100 \wedge \\ height(x) \geq 200 \wedge height(x) \leq 200 \wedge \\ font\_size(x) \geq 12 \wedge font\_size(x) \leq 12 \wedge \\ font\_type(x) = sans \\ \Rightarrow rdf{:}type(x, sioc{:}Post) \end{gathered} \tag{4.5}$$

Afterwards, the set of overfitting rules is iteratively reduced by grouping similar rules into more general ones, and by simplifying rules by generalizing conditions. A rule r^* is considered more general than rule r according to the following definition:

$$more\_general(r^*, r) \iff (lhs(r)(x_1, ..., x_n) \Rightarrow lhs(r^*)(x_1, ..., x_n)) \tag{4.6}$$

We use $rhs(r)$ to denote the right-hand side of a rule r and $lhs(r)$ for the left-hand side.

The process of generalization is shown in algorithm 1, which accepts a ruleset $\mathscr{R}$ and a set of HTML documents $\mathscr{D}$ on which perform the extractions. A

generalization operation is considered valid if the resulting ruleset produces a score as high as or higher than with the previous ruleset. Regarding the score function, we have used F-score, which will be defined in section 6.2.2, although other score function could be employed. The algorithm finishes when no more generalization operations can be performed, returning a new ruleset.

Algorithm 1 Rule generalization algorithm

```
1:  procedure GENERALIZE(ℛ, 𝒟)
2:      𝒯 ← ∅
3:      ℛ* ← ℛ
4:      repeat
5:          if SCORE(ℛ*, 𝒟) ≥ SCORE(ℛ, 𝒟) then
6:              ℛ ← ℛ*
7:          end if
8:          ℛ* ← ℛ
9:          for (r1, r2) ∈ ℛ × ℛ do
10:             if RHS(r1) = RHS(r2) ∧ ¬((r1, r2) ∈ 𝒯) then
11:                 𝒯 ← (r1, r2)
12:                 𝒫 ← {θi ∈ LHS(r1), θi(x, y) = parent(x, y)}
13:                 lhs* ← ⋀θi∈𝒫 θi
14:                 for (c1, c2) ∈ LHS(r1) × LHS(r2) do
15:                     if (c1 = (a(x) > th1)) ∧ (c2 = (a(x) > th2)) then
16:                         lhs* ← (lhs* ∧ (a(x) > min(th1, th2)))
17:                     else if (c1 = (a(x) < th1)) ∧ (c2 = (a(x) < th2)) then
18:                         lhs* ← (lhs* ∧ (a(x) < max(th1, th2)))
19:                     else if (c1 = (a(x) = v)) ∧ (c2 = (a(x) = v)) then
20:                         lhs* ← (lhs* ∧ (a(x) = v))
21:                     end if
22:                 end for
23:                 r* ← (lhs* ⇒ rhs(r1))
24:                 ℛ* ← ℛ \ {r1, r2} ∪ {r*}
25:                 break for
26:             end if
27:         end for
28:     until ℛ* = ℛ
29:     return ℛ
30: end procedure
```

This approach is similar to other machine learning techniques that perform rule pruning to reduce overfitting [Quinlan, 1993]. Our system thus requires a *building subset* out of the training dataset during the training phase. This building dataset is the one that is used to build the set of overfitting rules. Then, the whole

training dataset is used by the algorithm 1 to generalize the obtained rules.

As said, algorithm 1 progressively merges rules into new, more general ones. Lines 10–26 attempt to group two rules into a new one whenever two rules share the same structure. The requirement for this operation is that the two rules share the same right-hand side. Otherwise, the rules are not grouped and no new rule is built. We use $RHS(r)$ to denote the set of terms that appear on $rhs(r)$, and similarly $LHS(r)$ for the left-hand side. If two rules r_1 and r_2 are to be grouped, lines 14–22 state that for each of the terms c_1, from left-hand side of rule r_1, and c_2, from left-hand side of rule r_2, are combined into a more general one so that $c_1(x) \vee c_2(x) \Rightarrow c^*(x)$. This allows to produce a new rule r^* which is more general than the original rules r_1 and r_2. I.e., after combining r_1 and r_2, the following condition meets:

$$lhs(r_1)(x_1,...,x_n) \vee lhs(r_2)(x_1,...,x_n) \Rightarrow lhs(r^*)(x_1,...,x_n) \tag{4.7}$$

It can be proved that equation 4.7 implies $more\_general(r^*, r_1)$ and $more\_general(r^*, r_2)$.

An example of combining two rules is shown using the next ruleset:

$$\begin{gathered} r_1 = (width(x) > 100 \wedge width(x) < 300 \wedge font\_size(x) < 14 \wedge \\ parent(x,y) \wedge font\_size(y) > 16 \\ \Rightarrow rdf{:}type(x, sioc{:}Post) \wedge dc{:}title(x,y)) \end{gathered}$$

$$\begin{gathered} r_2 = (width(x) > 100 \wedge width(x) < 400 \wedge font\_size(x) < 14 \\ \Rightarrow rdf{:}type(x, sioc{:}Post)) \end{gathered} \tag{4.8}$$

$$\begin{gathered} r_3 = (width(x) > 250 \wedge width(x) < 400 \wedge font\_size(x) < 14 \wedge \\ parent(x,y) \wedge font\_size(y) > 14 \wedge font\_weight(y) > 500 \\ \Rightarrow rdf{:}type(x, sioc{:}Post) \wedge dc{:}title(x,y)) \end{gathered}$$

Only rules r_1 and r_3 can be combined because of sharing their right-hand sides. After combining term by term according to lines 14–22, the resulting rule r^* is:

$$\begin{gathered} r^* = (width(x) > 100 \wedge width(x) < 400 \wedge font\_size(x) < 14 \wedge \\ parent(x,y) \wedge font\_size(y) > 14 \\ \Rightarrow rdf{:}type(x, sioc{:}Post) \wedge dc{:}title(x,y)) \end{gathered} \tag{4.9}$$

The new rule is more general than the previous ones, as the condition from equation 4.7 is satisfied:

$$\begin{gathered}(width(x) > 100 \wedge width(x) < 300 \wedge font\_size(x) < 14 \wedge \\ parent(x,y) \wedge font\_size(y) > 16) \vee \\ (width(x) > 250 \wedge width(x) < 400 \wedge font\_size(x) < 14 \wedge \\ parent(x,y) \wedge font\_size(y) > 14 \wedge font\_weight(y) > 500) \\ \Rightarrow (width(x) > 100 \wedge width(x) < 400 \wedge font\_size(x) < 14 \wedge \\ parent(x,y) \wedge font\_size(y) > 14)\end{gathered} \tag{4.10}$$

Afterwards, the algorithm checks the new score with the resulting ruleset. If the score is as high as the previous one, the new ruleset is kept, otherwise the ruleset is rolled back. In all cases, a different pair of rules is tried in the next iteration (in order to achieve this, the already tried rule combinations are stored in set $\mathcal{T}$). The algorithm finishes when no more rules can be combined.

4.3.3 Wrapper conversion

When a scraper processes a visual mapping, it will obtain a set of RDF resources which are mapped to the fragments in a particular web resource. In our system, we will use this intermediate output to induct a wrapper with traditional wrapper induction techniques. This has the advantages explained next.

First, it improves the results of the visual patterns. Wrapper induction techniques require only a few correctly supervised samples to induct a wrapper [Anton, 2005]. When inducting a wrapper, all extracted data in a list can be extracted even if not all the samples are marked for extraction during the supervision process. This lets increase the recall of the visual patterns, as long as all news with a similar DOM tree will be selected whenever the visual extractor produces an acceptable amount of data as output.

Finally, wrappers are more lightweight, as they do not require visual features to be computed. The visual attributes enumerated above require a web browser to render the web resource and load CSSs and images. By converting the visual patterns into a wrapper, a web browser is not necessary anymore for using the mapping.

4.4 Conclusions

In this chapter, the induction of content discovery rules has been covered. An algorithm that performs induction of first-order logic rules to extract data from unstructured web resources has been described. Such system can be used to extract data from web sites with an unknown DOM tree structure, thanks to the fact that it is based on the visual features of the elements shown in the web browser. This allows extracting semantic information from unstructured web resources without external supervision, given a previous training stage.

Chapter 5

Service discovery

This chapter analyzes techniques for service discovery in the web. As said, developers are able to create new applications by composing already existing services from third-party vendors. However, the vast amount of choices, technologies and repositories can make discovering these services a tedious task. The chapter analyzes two approaches to perform discovery at the service level. Under one approach, services are discovered by the use of service discovery rules out of the knowledge provided by the underlying content layer. Under the second approach, services are found as content in web repositories and discovered as such. For that, the aspects behind service modelling are analysed and a component model is proposed.

5.1 Introduction

In the current Web, developers enjoy the availability of plenty of services, data feeds, widgets and other components that can be reused to build new web applications. This ecosystem of reusable web components comprise elements such as data feeds of various domains, telco services or desktop and mobile widgets. Additionally, there is a growing set of tools for the creation of mashups such as MyCocktail[1] or mashArt[2] that ease developers the combination of services for application construction. Also, Programmable Web[3], Yahoo Pipes[4] or Opera widgets[5] are examples of repositories that include services and widgets of many different kinds. They can be queried by users in order to search useful applications and services that they can reuse for mashup composition.

However, because of this mushrooming of web components and mashup platforms, developers face some difficulties when working in this development process of mashup construction. First, it is not immediate for a developer to find the most appropriate component for a mashup she is building, as there are many of them available and the information might be scattered across various repositories in the web. Second, components employ different standards and semantics, thus requiring some study of the documentation by the developer. And third, the components often need to be adapted for their usage by a specific mashup platform.

This chapter contributes to the solution of building semantic descriptions of services through two different approaches. Section 5.2.2 describes a novel approach called service probing, in which HTTP interactions are the input to service discovery rules, serving to identify features, which comprise service descriptions. Section 5.3 raises the fact that the information about several services is already available in web service registries such as Programmable Web and Yahoo Pipes. Therefore, services can be regarded as contents in this case, and be described using the same techniques outlined in section 4. For this, a component model is defined, which allows to build a metaregistry that aggregates services discovered as contents in the web.

[1] `http://www.ict-romulus.eu/web/mycocktail`
[2] `http://mashart.org`
[3] `http://www.programmableweb.com`
[4] `http://pipes.yahoo.com`
[5] `http://widgets.opera.com`

5.2 Services as REST resources

This section focuses on service discovery by regarding services as REST resources, which participate in HTTP interactions to provide value to the top, agent level of the discovery framework. For this task, a novel approach called service probing will be introduced after detailing the feature-oriented approach behind service descriptions, which was introduced in section 3.4.1.

5.2.1 Feature-Oriented descriptions

As said in chapter 3, the service model used for service discovery level applies ideas inspired in mixins, AOP, and FOP paradigms to semantic service description. These paradigms extend OOP by allowing the modelling of secondary concerns in an isolated way. FOP [Prehofer, 1997] is a composition model that allows refining classes through the definition of features, i.e., subclasses with core functionality. Mixins [Bracha and Cook, 1990], or abstract subclasses, are separate groups of methods that can be inserted into a class to override the original behaviour, but which cannot be instantiated on their own. Therefore, mixins serve to implement features inFOP [Apel et al., 2006]. FOP can be seen as a generalization of traditional class inheritance in OOP, and is used to develop the so-called software product lines, i.e. programs that provide different combinations of features [Lopez-Herrejon, 2005].

AOP [Elrad et al., 2001] similarly proposes separating concerns that cross-cut various classes or methods, with the logging aspect as the most popular example of a cross-cutting concern. Code from aspects is injected into specified join points in classes. This way, by usingAOP, logging commands can be inserted into appropriate join points in a class without changing the original code of the class.

AOP and FOP are different paradigms despite the existing similarity between them, as pointed out in [Lopez-Herrejon, 2005], so long as they propose different methods to combine code (code weaving vs. modification of inheritance chains). Some efforts try to combine the two approaches by introducing new concepts such as Multi Mixins, Aspectual Mixins, and Aspectual Mixin Layers [Apel et al., 2005].

Feature and aspect orientation have inspired other modelling approaches, such as Feature-Oriented Model-Driven Development (FOMDD) [Trujillo et al., 2007], in which models are created by composing features. Similarly, Role-Oriented Programming (ROP) [Steimann, 2000] or Subject-Oriented Programming (SOP)

[Harrison and Ossher, 1993] regard separation of object roles and the so-called subjective perceptions, respectively.

In the service level, the idea of separating features and concerns is employed to enable discovery at a feature level. A service description is a composition of features, allowing the reuse of feature descriptions in a similar way as features, aspects and abstract classes are reused in the previously mentioned paradigms. This modelling approach, and how is applied to service discovery rules, is described in this section.

In order to provide a mapping between a service's feature set and a service's formal description, the service level framework includes feature definitions. A feature definition $\mathscr{F}$ is a rule that maps a set of features f_1, ..., f_k to a set of conditions. It is activated when the set of features is present in the service's feature-oriented description, and produces a formal description. By aggregating all descriptions from the activated feature definitions, a service formal description can be obtained.

For example, given a service description F and features f_1 ("outputting a set of resources") and f_2 ("handling images"), some feature definitions can be the following:

$$f_1 \in F \Rightarrow |Output| > 1 \tag{5.1}$$

$$f_1 \in F \wedge f_2 \in F \Rightarrow \forall x(x \in Output \rightarrow image(x)) \tag{5.2}$$

Feature definition 5.1 formalizes feature f_1 by stating that the service's output cardinality has to be higher than one. Meanwhile, feature definition 5.2 formalizes that all output resources are images, and is applicable for services that (i) output a set of resources (as of feature f_1) and (ii) handle images (feature f_2). Therefore, the formal description of a service that is feature-oriented-described by f_1 and f_2 would be:

$$(|Output| > 1) \wedge \forall x(x \in Output \rightarrow image(x)) \tag{5.3}$$

It can be observed that feature definition 5.1 is activated by the presence of one feature in the feature-oriented description. Therefore, that definition serves to formalize one feature. However, feature definition 5.2 is activated by the presence of two features in the feature-oriented description.

Allowing definitions that are activated upon the presence of more than one feature might be regarded as unnecessary complexity. However, this versatility is justified. Consider feature f_3 as "The service stores a resource that is provided as input". When using f_2 with f_1, a postcondition should be set (i.e. "*multiple images*

Algorithm 2 Service probing algorithm

```
1:  procedure PROBE(I)
2:      S ← ∅
3:      repeat
4:          S* ← S
5:          for f ∈ ℱ \ S do
6:              valid ← false
7:              for interaction ∈ I do
8:                  if lhs(f)(i, S) then
9:                      valid ← true
10:                 end if
11:             end for
12:             if valid then
13:                 S* ← S ∪ {f}
14:             end if
15:         end for
16:     until S* = S
17:     return S
18: end procedure
```

will be returned"), but when used with feature f_3, a precondition should be set (i.e. "*an image has to be provided as input*"). This can be achieved by setting one definition for f_2 and f_1 (as shown in definition 5.2), and another definition for f_2 and f_3, as shown next:

$$f_2 \in F \wedge f_3 \in F \Rightarrow \forall x(x \in Input \rightarrow image(x)) \tag{5.4}$$

This serves to resolve the issue of feature interaction, already identified in FOP [Prehofer, 1997]. By describing features semantically, feature descriptions can be combined to produce a semantic service description.

5.2.2 Service probing

This section describes the approach of service probing, which is employed to discover services by applying discovery rules at the service level. The main idea behind the technique is to reason on the HTTP interactions performed on a REST resource to identify its functionalities, represented as features.

An HTTP interaction x is modelled as a method $method(x)$ (i.e. GET, $POST$, PUT, etc.), a set of inputs $Input(x)$, a set of outputs $Output(x)$, and a status $status(x)$, which, according to the HTTP specification [Fielding et al.,

Feature	Requires	Definition
`get`	-	$method(x) = GET \wedge status(x) \in \{200, 300, ..., 307\}$
`multiple`	`get`	$\lvert Output(x)\rvert > 1$
`filtered`	`get`	$t \in Input(x) \wedge y \in Output(x) \wedge ctag{:}tagged(y,t)$
`image`	`get`	$y \in Output(x) \wedge foaf{:}Image(y)$
`news`	`get`	$y \in Output(x) \wedge sioc{:}Post(y)$
`post`	-	$method(x) = POST \wedge status(x) \in [200, 400)$
`multiple`	`post`	$\lvert Input(x)\rvert > 1$
`image`	`post`	$y \in Input(x) \wedge foaf{:}Image(y)$
`news`	`post`	$y \in Input(x) \wedge sioc{:}Post(y)$

Table 5.1: Sample feature set for service probing

#	Method	Input
1	`get`	$ctag{:}Tag(t) \wedge rdfs{:}label(t, "beach")$
2	`get`	$ctag{:}Tag(t) \wedge rdfs{:}label(t, "sun")$
3	`get`	$ctag{:}Tag(t) \wedge rdfs{:}label(t, "holiday")$

Table 5.2: Requests in service probing sample

2009], indicates the result of the request. Service discovery rules use HTTP interactions as the input.

With a set of HTTP interactions, service discovery rules can be applied to determine the features that the service being discovered has. This set of interactions can be obtained after *probing* the service, i.e. querying the REST resource with sets of inputs and storing the resulting HTTP interactions.

The set of tentative inputs to be used on the service can be obtained by monitoring users via a browser plugin in order to collect full HTTP interactions. An automated approach after analyzing the domain of the web resource can be considered, but is out of the scope of the thesis and thus proposed as future work (see section 7).

Algorithm 2 shows the process of building a service feature-based description S out of a set of HTTP interactions I with a REST resource through service probing.

A service description S is iteratively built by selecting features out of the feature base $\mathscr{F}$. Lines 6–14 add a feature f to the service description whenever the left-hand side of a discovery rule is satisfactory. For that, the service description must already contain the features that the rule requires, and the interaction has to meet the conditions stated in the rule. If this applies to all the HTTP interactions,

#	Status	Output	
1	200 OK	$foaf{:}Image(y_1) \wedge$	$ctag{:}tagged(y_1,"tarifa") \wedge$
			$ctag{:}tagged(y_1,"beach") \wedge$
		$foaf{:}Image(y_2) \wedge$	$ctag{:}tagged(y_2,"ocean") \wedge$
			$ctag{:}tagged(y_2,"beach") \wedge$
			$ctag{:}tagged(y_2,"moon") \wedge$
		$foaf{:}Image(y_3) \wedge$	$ctag{:}tagged(y_3,"light") \wedge$
			$ctag{:}tagged(y_3,"woman") \wedge$
			$ctag{:}tagged(y_3,"beach")$
2	200 OK	$foaf{:}Image(y_1) \wedge$	$ctag{:}tagged(y_1,"summer") \wedge$
			$ctag{:}tagged(y_1,"sun") \wedge$
		$foaf{:}Image(y_2) \wedge$	$ctag{:}tagged(y_2,"summer") \wedge$
			$ctag{:}tagged(y_2,"sun") \wedge$
			$ctag{:}tagged(y_2,"film") \wedge$
			$ctag{:}tagged(y_2,"architecture") \wedge$
		$foaf{:}Image(y_3) \wedge$	$ctag{:}tagged(y_3,"sky") \wedge$
			$ctag{:}tagged(y_3,"sun") \wedge$
			$ctag{:}tagged(y_3,"backlight")$
3	200 OK	$foaf{:}Image(y_1) \wedge$	$ctag{:}tagged(y_1,"trip") \wedge$
			$ctag{:}tagged(y_1,"holiday") \wedge$
			$ctag{:}tagged(y_1,"japan") \wedge$
		$foaf{:}Image(y_2) \wedge$	$ctag{:}tagged(y_2,"trip") \wedge$
			$ctag{:}tagged(y_2,"travel") \wedge$
			$ctag{:}tagged(y_2,"holiday") \wedge$
		$foaf{:}Image(y_3) \wedge$	$ctag{:}tagged(y_3,"holiday") \wedge$
			$ctag{:}tagged(y_3,"europe") \wedge$
			$ctag{:}tagged(y_3,"croatia")$

Table 5.3: Responses in service probing sample

the feature is added to the description. Once there are no more definitions to try, or no modifications on the resulting description, the loop finishes and the description S is ready.

An example is given next. Let's suppose a particular feature set as base knowledge for the service level. The discovery rules for these features are shown in table 5.1.

Let's consider that a user uses Flickr's search form to find pictures about beaches, sun and holidays. The user employs the search terms `beach`, `sun`, and `holiday` on the input box, which results in different queries with their own HTTP methods, outputs, and statuses. These HTTP interactions involve pairs of requests and responses, which, according to the service discovery level (see section 3) can be translated into semantic contents thanks to content discovery rules.

Therefore, the raw output of the requests and responses of the search form

is converted into an RDF graph, and service discovery rules can be applied to identify features. Tables 5.2 and 5.3 show respectively the requests and responses of the HTTP interactions.

By checking the conditions for each service discovery rule, service features are identified, and a semantic, feature-oriented service description is built. Feature `post` is not identified because of the HTTP method used. Feature `get` is identified, and therefore other definitions with `get` are considered. Then, `multiple`, `filtered`, and `image` are identified, as their conditions fit the considered HTTP interactions. Finally, a `filtered image multiple get` service is discovered.

This section has shown how to discover services by probing with inputs, obtaining outputs and checking the conditions they meet. The feature-based approach allows to define service discovery rules that allow building service descriptions in this way.

5.3 Services as contents

Usually, services are documented in API pages that allow developers learn about their usage. These documentation pages are often registered in web sites such as Programmable Web or Yahoo Pipes. These kinds of web sites act as hubs in a similar fashion as news aggregators, allowing interested parties to be aware about services by accessing these centralized aggregators. Furthermore, repositories do not usually employ Semantic Web standards nor follow Linked Data principles, thus difficulting automatic processing, discovery or reasoning. This problem is thus similar to the one of semantic data availability. To sum up, services are served as contents and exposed in web service repositories, and as such the discovery methodology applied to contents can be applied to services under these conditions.

Therefore, this section covers how to perform service discovery when understanding services as contents. For this, an integrated metadirectory of web components for mashup composition is defined, which aggregates the knowledge offered in service registries by adding a semantic level. The metadirectory makes use of Linked Mashups Ontology (LiMOn), a component model that comprises useful information for querying web services and searching the most appropriate ones. Additionally, LiMOn reuses other underlying standards, such as WSMO [ESSI WSMO working group, 2004] or the W3C Widgets standard [Alario-Hoyos and Wilson, 2010], as low-level grounding description languages that allow web components to be readily executable. These descriptions are built automatically,

when possible, in a discovery phase that has allowed to populate the metadirectory with actual components from the web.

The section first analyzes the problem of choosing web components when developing mashups in SOAs and discusses how LiMOn fits in a framework of component selection. The metadirectory that makes use of LiMOn is described, as well as the approach that has been followed to populate the metadirectory with actual web components.

5.3.1 SOA domains

Mashup-Driven Development (MDD) proposes reusing web components to build new applications. These components vary from a REST service to a widget, gadget, portlet or even a web application that shares its information as a data feed. Therefore, when developing an application, developers can choose among a wide range of available components available to combine them and obtain a new working system. Then, developers face the problem of choosing the right component for the right task. First, the component needs to fit the functional requirements behind the tasks it has to perform in the newly constructed application. E.g. if a service for geolocation is seeked, a developer first needs to filter out all non-related services that do not deal with mapping services or geolocation. And second, the component needs to fit other non-functional requirements, such as trust in the company behind the component, or certain quality aspects that the component needs to meet.

Thus, developers need to search for the appropriate component according to some high-level needs they have. According to the type of component (API, service, widget...) the developers would have to check one registry or another (e.g. either a widget repository or some service registry). Also, depending on the features seeked in the component, some registries would be more appropriate than others (e.g. some registries might show information about semantics of the service and others not). And again, according to the features seeked, it would be necessary to query external sources to fill up the component information (e.g. it might be necessary to look up a components' vendor at Wikipedia in order to get an idea of the component's trust).

The Software Engineering Institute distinguishes several domains in the managament and development of SOA [Lewis and Smith, 2007]. These domains are business domain, engineering domain, operations domain and cross-cutting concerns. Each domain maps to an aspect which is relevant in component selection

when dealing with SOA:

- The business domain comprises all the consequences that a service orientation has on a given organization, application domain or context. This includes *business aspects* such as cost or legal issues when selecting a component for its reuse.
- The engineering domain deals with the service-oriented lifecycle. At the time of selecting a component for reuse, this domain would contain the *technical aspects* of a service, i.e. formats, interface, semantic descriptions, and so on.
- The operations domain deals with the operation, evaluation and optimization of service-oriented systems. Namely, in this domain the aspects that are involved when selecting a component will be *Quality of Service aspects*, which determines the evaluation of a component's performance.
- Cross-cutting concerns include aspects that are orthogonal to all the domains. The main aspects are *trust and social aspects*, which affect and determine other aspects in some way or another.

These aspects are detailed and discussed next, along with the references found in the literature.

Business aspects

Business aspects are any selection criteria that fall into the business domain of Service-Oriented Architectures. Whenever a decision aspect when selecting a service can have an impact on a given organization's structure, it is considered a business aspect to our understanding.

Cost is the most notable business aspect when selecting a service. Raj and Sasipraba [2010] understands cost as *the economic condition of using a service*, and includes it into a Quality of Service model for service selection. Similarly, Rehman et al. [2011], Zeng et al. [2009] and Li et al. [2010] define frameworks for comparison of cloud providers that also include cost as one of the selection aspects. Although some of these works use a broad definition of cost (involving non-monetary aspects as well), it can be considered a fundamental business aspect in component selection.

Legal restrictions are also a business aspect regarding component selection. Very often components are only usable under certain conditions on a reduced number

of countries, which affect the selection process depending on the company's activities and targets. Shimba [2010] considers legal issues as one challenge in cloud computing, enumerating the different difficulties that are encountered because of countries having different regulations and laws on the topic. The complexity behind this diverse regulation makes legal restrictions hard to model, which, as will be seen later, might be the reason why most component registries dismiss legal restrictions.

The *vendor* and the possible agreements with the consumer company may influence the decision of component selection. A company often agree to use other company's components under a certain domain. For example, Apple's mobile devices started to provide tight integration with Twitter although other microblogging services were available, after reaching an agreement that benefited both companies [Eaton, 2011]. Therefore, most times knowing the specific vendor that provides the service behind a web component is needed prior to taking the decision to use it. Thus, the vendor is another business aspect to consider regarding component selection.

Trust aspect

According to Amoroso et al. [1991], software trust is *the degree of confidence that exists that the software will be acceptable for "one's needs"*. This implies that, after a developer has been convinced about a software component's specifications, thanks to some documentation, trust would be the confidence that these specifications would be met over time. Related subaspects that have been identified are popularity, maturity, company trust and community trust.

Popularity is an indicative of success of a web component [Mileva et al., 2010]. Success can be understood in several ways, and different indicators can be used to measure it. A component that is widely used is considered succesful, while one with many bugs has lack of success. However, a component which lacks bug reports might also lack community support because of a lack in popularity. Thus, popularity is a combination of different indicators that convey an active usage of a big enough community of users. Popularity increases the so-called trusting-intention [Kutvonen, 2007], or the will to depend on another component with the involved risks. Therefore, popularity is a relevant metric inside the trust aspect.

Maturity is another software feature which increases trust. The topic of maturity is widely covered on the area of Open Source [Polancic et al., 2004]

because of the nature behind these kinds of projects – they usually follow an iterative growth, with frequent releases until they reach some point of maturity [Raymond, 1999]. The Capability Maturity Model (CMM) [Paulk, 1993] and the Open Source Software Maturity Model (OSSMM) [Golden, 2005] are models to improve the software process's maturity in companies for traditional software development and Open Source software development, respectively. These models emphasize the importance of seeking maturity in software.

Company trust is critical factor behind the global trust of component. A component is more trustworthy if a trustworthy company is behind it. Many factors are involved in comprising trust on a company, which can vary from company size to financial equity or customer service. Nguyen et al. [2006] identifies communication, cultural understanding and capabilities as the three top factors that determine trust on a software company. Similarly, a survey has for instance revealed that 75% of users perceive more trust on companies that use microblogging services such as Twitter [Gershberg, 2010]. All this reveals communication channels as key factors that determine company trust and thus the trust aspect.

Quality of Service aspects

Quality of Service in the Internet is traditionally regarded as *the combination of network-imposed delay, jitter, bandwidth and reliability* [Ferguson and Huston, 1998]. This is a typically network-level definition that can be extended to the application level by considering the metrics that a particular vendor offers for their commercial components. Hu et al. [2005] proposes a decision model of Quality of Service applied to Web Services that can be extended to components. They propose a model to select Web Services according to metrics of execution cost, execution time, reliability and availability. Similarly, Menasce and Almeida [2002] define Quality of Service as a combination of availability, security, response time and throughput issues. We will generalize these terms into availability, reliability and performance.

Availability is the percentage of time a web component is operating [Menasce, 2002]. When applying this definition to complex web components, this availability depends on the availability of several resources. I.e. in the case of a widget, both its assets (external scripts and HTML pages) and its services must be available for the widget's availability. Zhang and Zhang [2005] point out similar problems in the domain of mashups. We identify availability as an important aspect when

selecting a web component.

Reliability, as a general term in software, is *the probability of failure-free operation of a computer program for a specified time in a specified environment* [Musa et al., 1987]. This definition can be applied to Service-Oriented Architectures by considering web components as the software elements that are to provide failure-free operation. Zhang and Zhang [2005] state that a reliable web service *must exhibit correctness, fault-tolerance, testability, availability, performance and interoperability*. Those are a set of requirements that a service must keep in order to consider it failure-free. Other papers [Majer et al., 2009] consider as well the problem of reliability in more complex components, such as mashups, which reuse other services, thus depending on third-parties' reliability and availability.

We identify the *performance* aspect as a way to encompass execution times, responsiveness, and throughput issues, noted as important elements when regarding Quality of Service [Menasce and Almeida, 2002]. Depending on the nature of a web component, some metrics would make sense and others would not. For example, the concept of throughput cannot be applied to a web component such as widget (but it can be to a service), although its user interaction's responsivity can be measured in the same way as a web service's. These kinds of issues are regarded as performance aspects.

Technical aspects

The technical aspects behind selecting a web component involve all the issues related to component operation. Several component description standards such as WSDL [Christensen et al., 2001], WSMO [ESSI WSMO working group, 2004] or W3C widgets [Alario-Hoyos and Wilson, 2010] model the main characteristics that involve operation aspects for specific web component types such as services or widgets. The operation aspects these standards consider can be grouped into a few areas: interface, dependencies and cross-cutting concerns.

Interface comprises aspects such as conditions that are involved in the communication with the component. WSMO employs preconditions and postconditions to model a service's interface, and offers means to identify formats and protocols employed in the communication [Lara et al., 2004].

Dependencies include any requirement of external components. Especially, mashups [Majer et al., 2009] are components that are mainly built out of other components, such as web services, widgets or data feeds. Awareness about these dependencies can help to know about indirect requirements or usage restrictions

(if, e.g., a client-side mashup requires a geolocation service that is not available in the user's country).

Cross-cutting concerns involve non-functional technical aspects such as security, choreography issues, or required standards. Web Services often group these aspects in the so-called WS-* standards [Alonso, 2004]. Similarly, WSMO allows defining non-functional properties for a service in order to specify these kinds of aspects [Toma and Foxvog, 2006]. An interesting cross-cutting concern in SOA is discoverability, which measures *the extent to which the service, service consumers expect to look for, is easily and correctly found* [Choi and Kim, 2008]. It takes place if a component capabilities are published in one way or another. In order to allow web components to be found by developers, their capabilities need to be announced for an agent to discover them. A textual description of the component's functionality is a minimum requirement to make a component discoverable, although a semantic description allows automatic processing.

Support and coverage in existing repositories

In the web, we can find several repositories of components that can be reused for creating new applications. By taking a glance into these repositories, we can evaluate the support and coverage of the different aspects that we have identified for component selection.

A repository will be said to fully support an aspect if it provides appropriate and complete information related to that aspect. For example, Programmable Web provides a field to link a component to a WSDL file. A WSDL file is a Web Service standard description language that allows describing a service's interface, among other things. This allows components in Programmable Web to have full support to the interface aspect.

On the other hand, a repository provides full coverage of an aspect if all of its components make use of the supported fields for that aspect. For instance, Programmable Web supports WSDL annotated services by providing a field to reference a WSDL file. However, most APIs in Programmable Web do not make use of that field, hence resulting in low coverage of the interface aspect. The metric of coverage thus represents the actual degree of usage of a repository's capabilities.

Table 5.4 shows the analysis of support to every aspect on three repositories. The support that each repository gives to each aspect is marked from 0 points (no support) to 4 points (full support). We have selected three repositories that are both heterogeneous and popular. A short description of each of the considered

repositories is given next:

- Programmable Web is the most popular directory of mashups and APIs on the web. It is a collaborative directory where users can provide awareness of a mashup or an API. There are several fields that the users can fill up to provide information about a particular component. APIs such as Google Maps[6] and mashups such as Panoramio[7] have their own page where users add information on related APIs and mashups, or any other useful information.

- Yahoo Pipes is a repository of user-built mashups called "pipes". Each pipe is created using an editor developed by Yahoo, which allows users to create new data feeds out of existing ones. Then, pipes appear listed in Yahoo Pipes' web site, where users can find, and clone, other pipes and even reuse them to build new ones.

- Opera Widgets is a repository of widgets that are created by users. Opera does not provide an editor for this task, and simply allows users to upload their widgets and publish them in their web site. The web site then provides a browsing interface so that users can search widgets by category and load them into their browser.

As said, coverage is a complementary metric that reveals the actual degree of usage of each aspect. Fig. 5.1 illustrates the coverage for each repository and aspect. The region inside each graph represents the degree of coverage for each repository. It is worth noting that both Yahoo Pipes and Opera Widgets are strict repositories that require all fields to be filled for each component. Programmable Web, on the other hand, accepts optional information, such as the mentioned case of WSDL descriptions. As not all APIs in Programmable Web are linked to a WSDL file, this reduces the values in the technical axes.

5.3.2 Linked Mashups Ontology

In this section, we describe a model that integrates the properties and fields that are provided by current component repositories in the web. It is called the Linked Mashups Ontology (LiMOn), for its approach of bringing Linked Data to mashup-driven development.

[6] `http://maps.google.com`
[7] `http://panoramio.com`

Aspect	Subaspect	Programmable Web	Yahoo Pipes	Opera Widgets
Business	Cost	A field that plainly indicates whether or not there are usage fees is indicated (2 points)	All the "pipes" in the repository are free, so cost is implicit (4 points)	All the widgets in the repository are free, so cost is implicit (4 points)
	Legal issues	Links to commercial and free licenses (1 point)	All "pipes" share the same license (4 points)	All widgets share the same license (4 points)
	Vendor	Vendor is provided (4 points)	Author is shown (3 points)	Author is shown (3 points)
Trust	Popularity	Developers can rate APIs and mashups and the number of mashups that use an API is shown (4 points)	Number of "cloned pipes" are shown, which is an indicator of popularity (1 point)	Widget users can vote up and down widgets (3 points)
	Maturity	Addition date of a component is a naïve indicator of maturity (1 point)	Creation date of the "pipe" can be an indicator of maturity (2 points)	Addition date of a widget is a naïve indicator of maturity (1 point)
	Company trust	Vendor and home page are shown, but with no trust indicators (1 point)	Author is shown, but with no trust indicators (1 point)	Author is shown, but with no trust indicators (1 point)
QoS	Availability	No indicators (0 points)	No indicators (0 points)	No indicators (0 points)
	Reliability	No indicators (0 points)	No indicators (0 points)	No indicators (0 points)
	Performance	No indicators (0 points)	No indicators (0 points)	No indicators (0 points)
Technical	Interface	Yes, through a link to a WSDL service description (4 points)	Implicitly provided through an HTML form and a well-known uniform output for every "pipe" (4 points)	Yes, by sharing widgets the W3C widget standard (4 points)
	Dependencies	Mashups are connected to the APIs they use (4 points)	"Pipes" are connected with the feeds they use (4 points)	No information about dependencies (0 points)
	Cross-cutting concerns	Information about SSL usage, category, tags, plus the information available in WSDL (4 points)	Tags and a textual description are provided to categorize a pipe (2 points)	Only a taxonomy and a textual description is provided (2 points)

Table 5.4: Repositories' support to aspects

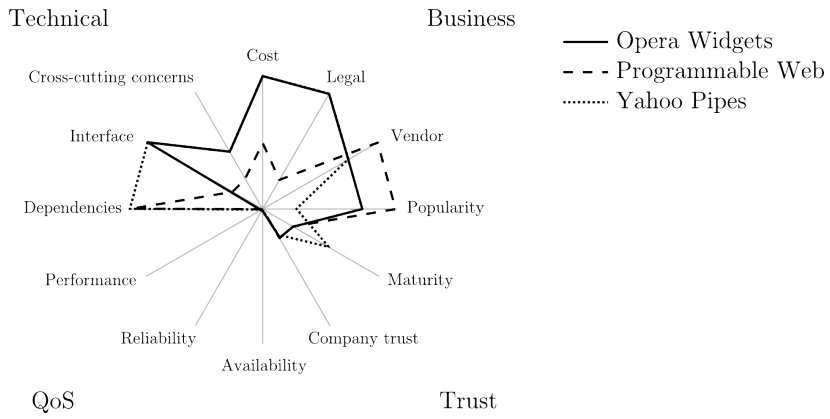

Figure 5.1: Repositories' coverage of aspects

With these considerations in mind, we have defined the model presented in Fig. 5.2. Regarding the technical aspect that was introduced in section 4.2, the properties listed next have been included in the model.

- A set of properties allow to cover interface aspects. The property *description* allows linking to a lower-level component description, such as WSMO, W3C widgets or WSDL, depending on the nature of the component. The property *endpoint* allows to link to the particular URL where the component runs. Also, the properties of *dataFormat* and *protocol* allow to specify how the data, if any, is exchanged with the component.

- The property *uses* is employed to link to reused components. For example, it can be used to indicate which services or data feeds a mashup reuses.

- Some properties address cross-cutting concerns. The property *clientInstallRequired* indicates whether or not the web component requires an additional component installed client-side to work. The property *example* allows to reference examples of usage of the component's API. The properties *tag* and *category* allow to link to tags and categories, respectively, that represent the functionality of the component. The property *api* links to the specification of the component's programming interface. Finally, the properties *authentication* and *sslSupport* allow to specify how the security over component's data transport is performed.

The trust aspect comprises popularity and company trust issues, and includes the properties listed next:

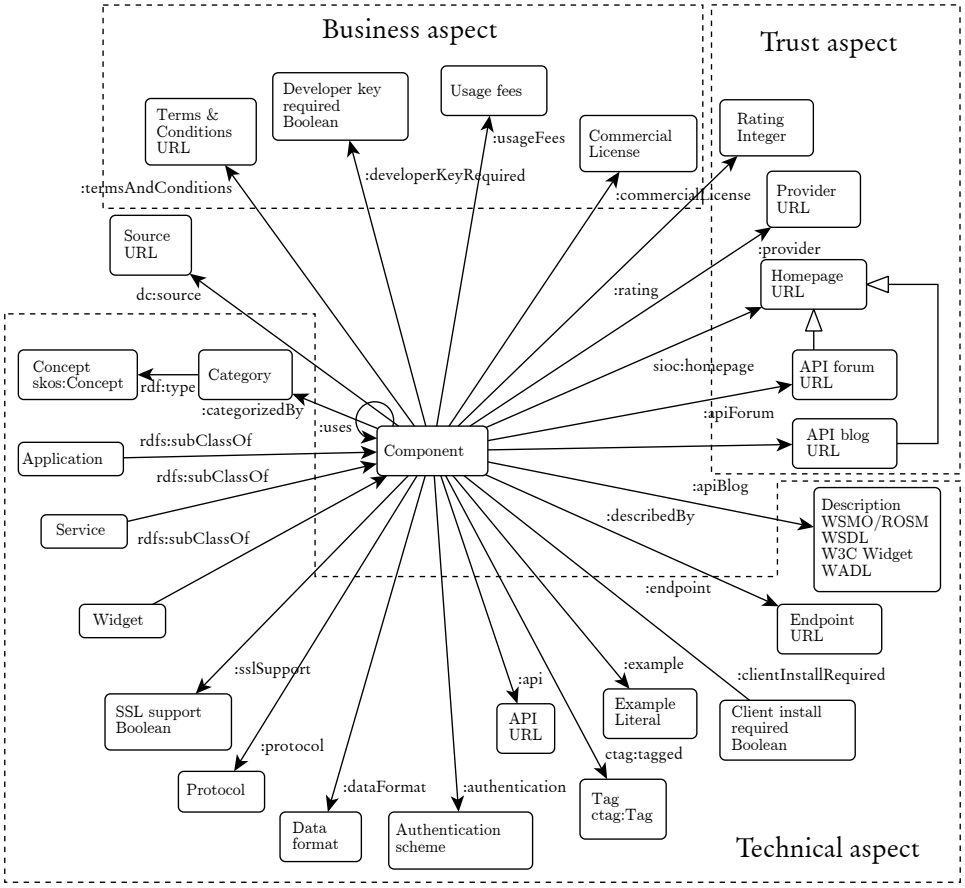

Figure 5.2: Linked Mashups Ontology

- The *rating* property serves as an indicator of popularity. It represents the rating made by users in repositories to reflect their degree of satisfaction with a particular component.

- The properties of *apiForum* and *apiBlog* fall into the company trust by providing means to reference support facilities (i.e. forums and blogs) that the vendor provides to component users. Also, the property *provider* allows to identify the vendor of the component, for any company trust issues involved.

The business aspect comprises costs or legal issues, and is covered by the model through the following properties:

- The property *usageFees* is a cost aspect property that link to any cost required

when using the component.

- Regarding legal issues, the property *termsAndConditions* allows linking to a document that informs about the conditions of usage of the component. The property *commercialLicense* links to a commercial license for the usage of the component, if any. Finally, the property *developerKeyRequired* indicates whether or not the component requires creating a developer account prior to its usage.

- The property *provider* serves to identify the vendor of the component for any business issues involved.

Additionally, the source of a component is included as a property. In next sections, the model will be used to build a metadirectory. This makes it useful to reference where the component was obtained from, thus requiring a property to link it to the source repository. Also, regarding the Quality of Service aspect, no information was found in any repository, so no field was included in the model.

In total, the component repositories of Yahoo Pipes, Programmable Web, Opera Widgets, iGoogle Gadgets[8], AppStore[9], Android Market[10], and Ohloh[11] were analysed to identify relevant properties for the model.

Fig. 5.3 shows the connections between the component model and the ontologies that have been reused, illustrating how these links can be exploited thanks to already existing tools. Such tools generally consist of ways of exploiting the Linked Data graph, either by allowing the identification of new relations between resources or by providing ways to visualize the data.

In chapter 6, the evaluation of applying this model to discover services from the web is performed.

5.4 Conclusions

Through this chapter two different approaches to service discovery have been covered. First, strictly on the service level of the discovery framework, the discovery process has been shown to be possible thanks to a novel technique called service probing. With this approach, service discovery rules are applied to sets of

[8] `http://www.google.com/ig/directory`
[9] `http://itunes.apple.com/de/genre/ios/id36`
[10] `https://market.android.com`
[11] `http://www.ohloh.net`

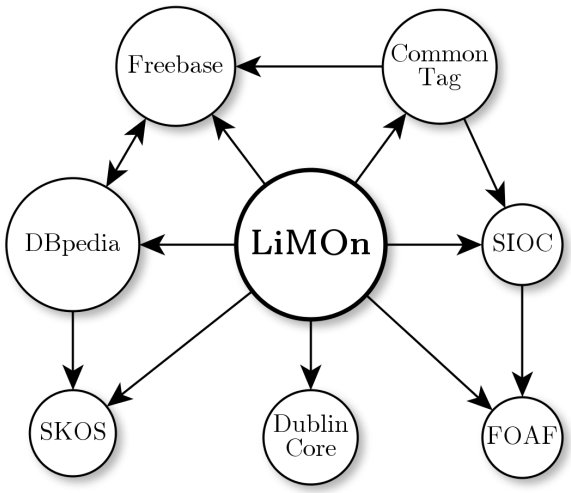

Figure 5.3: Connections between LiMOn and other ontologies

HTTP interactions to identify service features and therefore discover services. The second approach considers the problem where services are already referenced in the contents of web sites, which is the usual case in web registries and repositories such as Programmable Web and Yahoo Pipes web sites. In this case, the same methodology as in content discovery can be applied, and a component model, LiMOn has been defined for performing content-level discovery of services.

Chapter 6

Evaluation

In this chapter, the evaluation of the discovery framework is described. The framework and its different layers have been applied to several scenarios. The content layer was evaluated in scenarios of security knowledge extraction, news discovery, and idea management. The service layer was applied to mashup development. The full framework is used in a scenario of electronic newspapers. An intelligent agent crawls the web for related news, and uses services and visits links automatically according to its goal. The scenario illustrates how the discovery is achieved at the different levels and how the use of semantics help to implement agents that perform high-level tasks. The chapter therefore summarizes the evaluations performed to validate the proposed framework.

6.1 Introduction

This chapter describes the evaluation of the different contributions of the thesis. The discovery framework has been evaluated on a variety of scenarios. These scenarios are the result of the definition of different case studies in several research projects. The main projects are listed next.

ROMULUS (FP7-ICT-2007-1) is a European funded project which researches into MDD and agile development by defining a unified framework for mashup development. The Spanish funded project Java sobre Ruedas (FIT-350401-2007-8) similarly deals with MDD and agile development, and thesis' contributions have been also applied on it. The framework was applied at the service level to provide integration of REST services in mashups.

Contenidos a la Carta (TSI-020501-2008-114) is a Spanish funded project which researches into the improvement of news processing by the use of mashup development techniques. The contributions on content discovery from the thesis were applied extensively in this project to discover relevant news and build semantic descriptions for content combination.

OMELETTE (FP7-ICT-2009-5) is a European funded project which researches into the definition of an open framework for the construction and execution of telco mashups. The service discovery level of framework was applied extensively on this project to provide mashup component discovery in order to keep the database of mashup components of the platform up to date.

Other projects where the discovery framework was used are THOFU project, which deals with innovation in hotels and tourism, Resulta project (TSI-020301-2009-31), which deals with the improvement of business collaboration in a web 2.0 basis, and VulneraNET project (TSI-020302-2009-64), which deals with the provision of software security knowledge. In these cases, the thesis' contributions were applied in order to support other research lines.

The chapter's structure follows the different levels of the framework. Section 6.2 covers content discovery evaluation. Section 6.3 describes the evaluation of service discovery. Finally, section 6.4 describes the evaluation of the agent level of the framework.

6.2 Content discovery

This section covers the evaluation of the content discovery. A software system called Scrappy[1] was built to perform content discovery. It is an Open Source Semantic scraper that uses the Scraping Ontology for its mappings and implements the induction algorithm 1 for building generalizable mappings automatically.

6.2.1 Scraping Ontology

This section covers the evaluation of the Scraping Ontology. The Scraping Ontology was used to manually build content discovery rules in several scenarios.

In projects Resulta and THOFU, the Scraping Ontology was employed to validate GI2MO ontology [Westerski et al., 2010], an ontology for Idea Management [Westerski and Iglesias, 2012] which was employed in these projects. Some web sites with ideas, such as Ubuntu Brainstorm[2], were mined in order to obtain about 28,000 of semantically-annotated ideas for further processing [Poveda-Cardona, 2011].

In project Contenidos a la Carta, the Scraping Ontology was employed to mine eight electronic newspapers and combine data from different sources. The ontology served to extract information that is not present in content aggregators like Really Simple Syndication (RSS) feeds, such as location, comments or the full body of news.

In project VulneraNET, the Scraping Ontology was employed to automatically build an ontology of software security knowledge by mapping the contents of The Open Web Application Security Project (OWASP)[3] onto semantic entities. Content was discovered after crawling the website with the defined content discovery rules.

Project OMELETTE uses the Scraping Ontology to discover services, and is thus detailed in section 6.3.

The variety of scenarios provides an idea of the flexibility of the ontology to create mappings for extracting information from the web after applying the mappings and using Scrappy to extract the data.

[1] `http://github.com/josei/scrappy`
[2] `http://brainstorm.ubuntu.com`
[3] `http://owasp.org`

6.2.2 Automatic rule induction

This section covers how the rule induction algorithm from section 4.3.2 was employed to generate content discovery rules.

The system has been evaluated on a set of web pages. It has been trained to extract news posts with title, description and image by using FOAF [Brickley and Miller, 2000], DC and SIOC ontologies, chosen because of their high adoption and popularity. An example of extracted piece of news in RDF is the following:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:foaf="http://xmlns.com/foaf/spec/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:sioc="http://rdfs.org/sioc/ns#">
  <sioc:Post
    rdf:about=
    "http://abc.es/20110629/internacional/abci-bolivia-coca-201106291637.html">
    <dc:description>La Convencion de Viena considera la hoja de coca un estupefaciente,
      pero su masticado es una practica ancestral de los indigenas del pais
      sudamericano
    </dc:description>
    <dc:title>Bolivia denunciara la convencion de la ONU que prohibe el masticado de
      coca
    </dc:title>
    <foaf:depiction
      rdf:resource="http://www.abc.es/Media/201106/29/10557169--229x229.jpg"/>
  </sioc:Post>
</rdf:RDF/>
```

In the evaluation, a set of experiments are run to check the performance of the solution. First, the metrics that will be obtained out of each experiment are defined. Then, the training and testing datasets which have been used are described, and, finally, the results are presented and discussed.

Evaluation metrics

In order to evaluate the algorithm, a set of metrics will be calculated out of each test. Typically, recall and precision are the most common metrics used in information extraction, and they will be used in the evaluation along with F-score, a combined metric of precision and recall. These metrics are defined next.

Given a single extraction of a set of data out of a web resource, let n^+ be the number of triples that were extracted right, let n be the number of triples that were extracted, and let N be the number of triples that should have been extracted.

With these variables, the following formulae for *precision* and *recall* is obtained:

$$precision = \frac{n^+}{n} \tag{6.1}$$

$$recall = \frac{n^+}{N} \tag{6.2}$$

Precision and recall are separate metrics that provide an idea of the performance of an information extraction or retrieval test. Precision indicates the ratio of results that are correct, while recall indicates the ratio of correct results that are extracted.

In order to have a global indicator that combines both metrics, *F-score* metric is defined. F-score is defined as the harmonic mean of precision and recall, which lets us write:

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{6.3}$$

As of section 4.2, it is desirable to measure the robustness and generalization of the system. To do so, the testing data is splitted into a robustness test data and a generalization test data, so robustness and generalization can be defined as follows.

$Robustness$ is defined as the mean of F-scores on a set $\mathcal{S}_R$ of DOM-altered web resources r_i:

$$robustness = \frac{\sum_{r_i \in \mathcal{S}_R} F(r_i)}{|\mathcal{S}_R|} \tag{6.4}$$

A DOM-altered web resource is a web resource taken from the training dataset which has been subject to changes in its DOM tree. This makes traditional wrapper-based approaches to fail, as they base the extraction on a particular structure of the DOM tree. Examples of variations performed in the DOM tree can be renaming CSS classes, removing parent nodes, or relabelling HTML nodes. These variations affect only the internal structure of the document, while the visual aspect experiences less changes.

$Generalization$ is defined as the mean of F-scores on a testing set $\mathcal{S}_T$ of web resources r_i which belong to web sites that were not used at training time.

$$generalization = \frac{\sum_{r_i \in \mathcal{S}_T} F(r_i)}{|\mathcal{S}_T|} \tag{6.5}$$

Table 6.1: Training dataset

	elpais.es	abc.es
sioc:Post	326	122
dc:title	343	126
dc:description	343	123
foaf:depiction	175	56
Total triples	1193	427

Because these resources have completely different DOM trees from the resources used in the training phase, wrapper-based techniques cannot be applied, as they would require a new wrapper to be constructed. The approach is based on visual features, so it will be able to extract data, as for usability reasons web sites often share similar visual aspects.

Evaluation datasets

The datasets used for the evaluation are home pages of three Spanish newspapers: Abc[4], El País[5] and El Mundo[6]. These web sites have some visual aspects in common and therefore comprise a suitable dataset for the evaluation.

Wrappers were built manually to obtain the supervised data of these web sites. Sample home pages from different days for each newspaper were selected and a set of RDF triples were extracted out of them. Table 6.1 summarizes the data in the training dataset, while table 6.2 shows some samples with their attributes and classes. Table 6.3 summarizes the testing datasets.

Abc and El País home pages are used as training datasets, while El Mundo is used as testing dataset for the generalization test. Regarding the robustness test, El País newspaper performed a layout redesign on 23^{rd} May, 2011, as of Spanish elections held the day before, in order to better present the elections' results. This affected the performance of the manually constructed wrapper, and therefore makes it an interesting testing sample for the robustness test.

Results and discussion

Table 6.4 shows the results of the generalization test. As the testing samples belong to a news site, they share some visual aspects with the training data, so the

[4] `http://www.abc.es`
[5] `http://www.elpais.es`
[6] `http://www.elmundo.es`

Table 6.2: Training samples

	X	Y	Width	Height	Font size	Font type	...	Triple
x_1	118	5552	310	243	N/A	N/A	...	$rdf{:}type(x_1, sioc{:}Post)$
x_2	119	5558	310	26	18	serif	...	$dc{:}title(x_1, x_2)$
x_3	118	5736	310	28	16	sans-serif	...	$dc{:}description(x_1, x_3)$
x_4	787	332	300	703	N/A	N/A	...	$rdf{:}type(x_4, sioc{:}Post)$
x_5	787	507	284	53	22	sans-serif	...	$dc{:}title(x_4, x_5)$
x_6	787	569	300	34	12	sans-serif	...	$dc{:}description(x4, x_6)$
x_7	114	1247	390	301	N/A	N/A	...	$rdf{:}type(x_7, sioc{:}Post)$
x_8	114	1474	299	26	22	sans-serif	...	$dc{:}title(x_7, x_8)$
x_9	114	1509	390	34	12	sans-serif	...	$dc{:}description(x_7, x_9)$
x_{10}	114	1258	390	194	N/A	N/A	...	$foaf{:}depiction(x_7, x_{10})$

Table 6.3: Testing datasets

	Robustness test elpais.es	Generalization test elmundo.es
sioc:Post	79	529
dc:title	79	546
dc:description	79	545
foaf:depiction	39	223
Total triples	276	1843

Table 6.4: Evaluation of generalization

	Wrapper	Rules
Triples	–	1843
Extracted triples	–	1423
Correct triples	–	1325
sioc:Post precision	–	98.07%
sioc:Post recall	–	76.94%
sioc:Post F-score	–	86.23%
dc:title precision	–	98.13%
dc:title recall	–	76.74%
dc:title F-score	–	86.13%
dc:description precision	–	83.60%
dc:description recall	–	76.70%
dc:description F-score	–	80.00%
foaf:depiction precision	–	100.00%
foaf:depiction recall	–	36.32%
foaf:depiction F-score	–	53.29%
General precision	–	**93.11%**
General recall	–	**71.89%**
General F-score	–	**81.14%**

system managed to extract most pieces of news right. Regarding precision, the system failed to extract properly some pieces of news that are shown in figure 6.1. According to the learned patterns, the top text in a piece of news should represent the title and the lowest one the news description, although in this case the top text is just a news category. Also, the system could not extract some pieces of news in the middle column, as their size is smaller than the pieces of news that were used to learn the patterns, which affected the results of the recall metric.

In the case of the technique of wrapper induction, it requires a new wrapper to be constructed for each new site. Therefore, generalization results do not include wrapper induction as long as it does not have generalization capabilities.

Table 6.5 shows the results of robustness test. The system shows top precision and very high recall. The high precision is achieved because all the news that appear in the web site have a similar one whose pattern was already learned in the training phase. Regarding recall, the system managed to extract news that were published under a new layout, while the manually built wrapper failed to achieve so.

The tests prove the robustness of the approach, which makes it a suitable tool

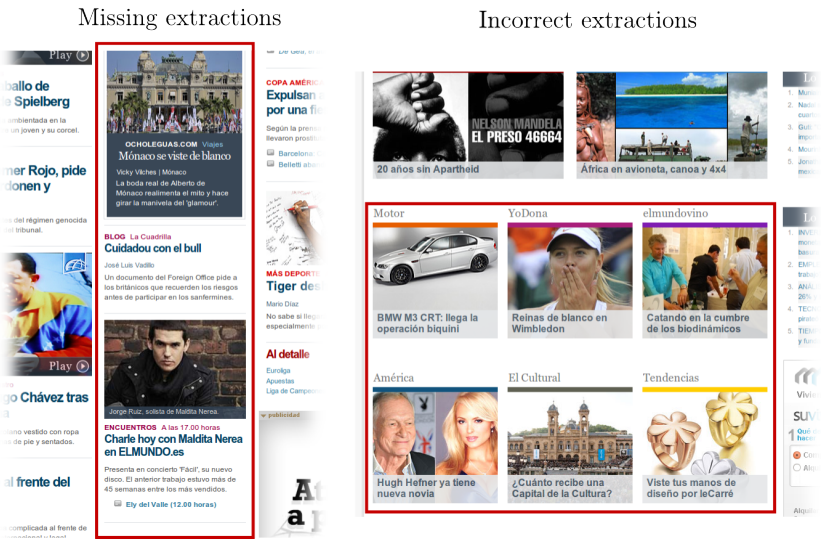

Figure 6.1: Extraction errors on a new web site

Table 6.5: Evaluation of robustness

	Wrapper	Rules
Triples	276	276
Extracted triples	196	238
Correct triples	196	238
sioc:Post precision	100.00%	100.00%
sioc:Post recall	68.35%	86.08%
sioc:Post F-score	81.20%	92.52%
dc:title precision	100%	100.00%
dc:title recall	67.50%	86.25%
dc:title F-score	80.60%	92.62%
dc:description precision	100.00%	100.00%
dc:description recall	67.50%	86.25%
dc:description F-score	80.60%	92.62%
foaf:depiction precision	100.00%	100.00%
foaf:depiction recall	91.89%	86.49%
foaf:depiction F-score	95.77%	92.75%
General precision	100.00%	**100.00%**
General recall	71.01%	**86.23%**
General F-score	83.05%	**92.61%**

for automatic maintenance of wrappers. Additionally, the system shows good generalization capabilities, which turns it into a useful tool for the automatic, unsupervised generation of wrappers in unforeseen web sites. The evaluation has been performed on a specific domain. In a different domain, it is expected that the rules would extract those HTML fragments that resemble pieces of news according to the evaluation performed. Mismatches would only happen in those cases where fragments visually appear to be pieces of news but which are not because of their contents.

6.3 Service discovery

This section describes the evaluation of the service level of the discovery framework. The approach of using features to describe services is evaluated on the domain of picture search to build a metasearch service. Also, service discovery is evaluated by building a metadirectory that aggregates the services available on several different repositories from the web.

6.3.1 Service probing

As said, feature-oriented descriptions attempt to simplify the process of defining service descriptions to favour automatic service consumption in the Semantic Web. This section defines a vocabulary for describing services that perform retrieval operations, i.e. search services, and discovers a set of service using service probing. In order to narrow the scope of the task, only image search services are considered. These kinds of services attempt to fulfil the user's goal of finding a particular kind of image on the web. This work was performed to provide a baseline of search services for the mashup platform of ROMULUS project.

In order to perform the task of discovering a set of image search services, the next steps are followed:

1. Identify popular image search services.
2. Collect features across the considered services.
3. Model features through feature definitions.
4. Adapt services to fit the modelled interface.
5. Discover service features for each service using service probing.

Service modelling

Google Images, Panoramio, Bing Images and Flickr search are the search services that are analyzed and modelled. Other image search services are available, but the decision to choose this set is based on the fact that:

1. Flickr offers capabilities that are not present on the other services, such as tag search.

2. Panoramio offers additional capabilities that are not present on the rest of services, such as search by location.

3. Google Images and Bing Images have similar capabilities, but offer a different interface. This fact could impact on the vocabulary.

After identifying the services that are considered for the modelling task, raw features from these services are collected. By observing each of these services, it can be observed that some features define the obvious behaviour of a search service and are enabled by default, while others require additional configuration in the service, often accessible through an "advanced search" option. Therefore, for this task, it was necessary to access "advanced search" and help pages of the services in order to discover their capabilities.

It is immediate to identify raw features such as "the service returns a set of images as search result". However, this raw feature can be split into finer-grained features, such as "returning a set of results", "returning an image" and "performing a retrieval operation". The combination of these three features matches the aforementioned raw feature.

Table 6.6 summarizes the features that have been identified in the considered services. The first column shows raw features that were identified in the services, while the next column represents the specific finer-grain features that are produced. The third column is a shorthand term that represents the feature.

The next step is modelling the features, so that feature discovery can be performed and an extended service description can be produced out of a plain list of features. This is done by defining features through service discovery rules. An example is the fact of outputting images: feature `image` and feature `get` should set the condition that the service's output has to be an image, which can be expressed

Raw feature	Feature	Term
A set of images is returned to the user as search result	Retrieval operation performed	`get`
	Multiple resources involved	`multiple`
	Resource represents an image	`image`
Images are filtered according to a location	Resource is filtered by location	`location-filtered`
Images are indexed from anywhere in the web	Resource cannot belong to another domain	`local`
Images are summarized and have a thumbnail	Resource is summarized	`summarized`
	Resource has a title	`titled`
Images can be searched by keywords	Resource is filtered by given keywords	`keyword-filtered`
Images can be searched by tags	Resource is filtered by given tags	`tag-filtered`
Images can be searched by dimensions	Resource is filtered by dimensions	`size-filtered`
Images can be searched by creation or taking date	Resource is filtered by creation date	`creation-filtered`
	Resource is filtered by publishing date	`publishing-filtered`
Images can be selected by license	Resource is filtered by license	`license-filtered`
Images can be searched by colour and content	Resource is filtered by colour	`colour-filtered`
	Resource is filtered by content	`content-filtered`

Table 6.6: Analysis of detailed features out of raw ones

through the following discovery rule:

$$\begin{gathered} y \in Output(x) \wedge sioc{:}Image(y) \Rightarrow \\ ms{:}has\_feature(x, image) \wedge ms{:}has\_feature(x, get)) \end{gathered} \tag{6.6}$$

This feature definition solves the feature interaction between `image` and `get`.

Therefore, for each of the identified features, a set of feature definitions that consider the possible feature interactions are built. The identified feature definitions are shown in table 6.7.

The first column shows the features that are involved in each definition, and the next two columns show the preconditions and postconditions. These feature definitions allow producing an extended service description when combining different features. This service description can be processed to perform operations

Feature	Conditions
`get`	$method(x) = GET \wedge status(x) = 200$
`image`	$y \in Output(x) \wedge feature(x, get)$
`multiple`	$\lvert Output(x)\rvert > 1 \wedge feature(x, get)$
`summarized`	$y \in Output(x) \wedge dc{:}description(y, z) \wedge feature(x, get)$
`keyword-filtered`	$y \in Input(x) \wedge rdf{:}type(y, rdf{:}Literal) \wedge$ $z \in Output(x) \wedge dc{:}subject(z, y) \wedge feature(x, get)$
`tag-filtered`	$y \in Input(x) \wedge ctag{:}Tag(y) \wedge$ $z \in Output(x) \wedge ctag{:}tagged(z, y) \wedge feature(x, get)$
`local`	$y \in Output(x) \wedge domain(y) = domain(x)) \wedge$ $feature(x, get)$
`titled`	$y \in Output(x) \wedge dc{:}title(y, z) \wedge feature(x, get)$
`size-filtered`	$y \in Input(x) \wedge pic{:}Size(y) \wedge z \in Output(x) \wedge$ $pic{:}size(z, y) \wedge feature(x, image) \wedge feature(x, get)$
`creation-filtered`	$y \in Input(x) \wedge rdf{:}type(y, pic{:}MinCreation) \wedge$ $z \in Output(x) \wedge dc{:}dateSubmitted(z, j) \wedge y < j \wedge$ $feature(x, get)$
`publishing-filtered`	$y \in Input(x) \wedge rdf{:}type(y, pic{:}MinPublishing) \wedge$ $z \in Output(x) \wedge dc{:}dateAccepted(z, j) \wedge y < j \wedge$ $feature(x, get)$
`license-filtered`	$y \in Input(x) \wedge rdf{:}type(y, pic{:}License) \wedge$ $z \in Output(x) \wedge dc{:}license(x, z) \wedge feature(x, get)$
`location-filtered`	$y \in Input(x) \wedge rdf{:}type(y, loc{:}Location) \wedge$ $z \in Output(x) \wedge loc{:}inLocation(z, y) \wedge$ $feature(x, get)$
`content-filtered`	$y \in Input(x) \wedge rdf{:}type(y, pic{:}Content) \wedge$ $z \in Output(x) \wedge pic{:}content(z, y) \wedge$ $feature(x, get)$
`colour-filtered`	$y \in Input(x) \wedge rdf{:}type(y, pic{:}Colour) \wedge$ $z \in Output(x) \wedge pic{:}colour(z, y) \wedge$ $feature(x, get)$

Table 6.7: Specification of features

such as automatic validation (validating conditions), automatic discovery of the semantics of inputs, or automatic user interface generation, as will be seen later on.

A system that allows wrapping existing services into services that fit a specific interface was implemented. In the proposed vocabulary, there are assumptions such as input types and semantics. In order for the services to fit this interface, an adaptation has to be performed. For each service, content discovery rules are defined to transform information into the specific model. This lets using the existing services with the interface that has been defined in the vocabulary.

#	Method	Input
1	`get`	$rdfs{:}label(x_1,"guitar") \wedge rdf{:}type(x_2, pic:BigSize)$
2	`get`	$rdfs{:}label(x_1,"rock")$
3	`get`	$rdfs{:}label(x_1,"blues") \wedge rdf{:}type(x_2, ctag{:}Tag) \wedge$ $rdfs{:}label(x_2,"eric\ clapton") \wedge$ $rdf{:}type(x_3, pic:RedColour)$

Table 6.8: Sample requests used for service probing

Service probing

Once feature definitions are defined and wrappers for each service are created, the service probing algorithm can be applied to discover service descriptions. For such task, according to 5.2.2, a set of input requests for the services are generated by monitoring the user's execution of such services. In other words, a user performs sample searches entering keywords, filtering by the available options of the services, and these interactions are recorded as samples. A set of sample requests obtained and used as input for service probing is shown in table 6.8. The associated outputs for these requests when employing Google Images are shown in table 6.9.

When executing the algorithm, discovery rules are applied to the sample HTTP interactions and features are identified. From the supplied subset of samples shown in tables 6.8 and 6.9, the features get, multiple, image, keyword-filtered, size-filtered and summarized were identified. Although tags for filtering are provided on the third request sample, Google Images does not respond to such inputs, and thus feature `tag-filtered` is not identified, as expected.

An issue with certain features is the lack of semantic annotations on the output that reveal some properties such as colour of the images or the type of content (e.g. is it a picture of faces, or clipart?). This requires additional manual annotation of outputs in order to obtain suitable results when running the service probing algorithm.

Table 6.10 shows the discovered services' capabilities, i.e. the presence of the identified features on the considered services after running the service probing algorithm. As anticipated, Google Images and Bing Images share the same capabilities, while Flickr and Panoramio services serve to enrich the vocabulary in order to cover a wider set of features.

Each service can be executed through a user interface that is generated automatically by analysing the feature discovery rules. Also, a description of the

#	Status	Output	
1	200 OK	$foaf{:}Image(y_1) \wedge$	$ctag{:}tagged(y_1,"guitar") \wedge$
			$dc{:}description(y_1,"Guitar\ wallpaper...") \wedge$
			$pic{:}size(y_1, pic:BigSize) \wedge$
		$foaf{:}Image(y_2) \wedge$	$ctag{:}tagged(y_2,"guitar") \wedge$
			$dc{:}description(y_2,"Wireless\ guitar") \wedge$
			$pic{:}size(y_2, pic:BigSize) \wedge$
		$foaf{:}Image(y_3) \wedge$	$ctag{:}tagged(y_3,"guitar") \wedge$
			$dc{:}description(y_3,"Electric\ guitar...") \wedge$
			$pic{:}size(y_3, pic:BigSize) \wedge$
		$foaf{:}Image(y_4) \wedge$	$ctag{:}tagged(y_4,"guitar") \wedge$
			$dc{:}description(y_4,"Heritage\ cherry...") \wedge$
			$pic{:}size(y_4, pic:BigSize) \wedge$
		$foaf{:}Image(y_5) \wedge$	$ctag{:}tagged(y_5,"guitar") \wedge$
			$dc{:}description(y_5,"How\ to\ play...") \wedge$
			$pic{:}size(y_5, pic:BigSize) \wedge$
		...	
2	200 OK	$foaf{:}Image(y_1) \wedge$	$dc{:}subject(y_1,"rock") \wedge$
			$dc{:}description(y_1,"During\ the\ times...") \wedge$
		$foaf{:}Image(y_2) \wedge$	$dc{:}subject(y_2,"rock") \wedge$
			$dc{:}description(y_2,"Too\ old\ to\ rock") \wedge$
		$foaf{:}Image(y_3) \wedge$	$dc{:}subject(y_3,"rock") \wedge$
			$dc{:}description(y_3,"Rocks\ and\ soils") \wedge$
		$foaf{:}Image(y_4) \wedge$	$dc{:}subject(y_4,"rock") \wedge$
			$dc{:}description(y_4,"Rock\ saved\ my\ life...") \wedge$
		$foaf{:}Image(y_5) \wedge$	$dc{:}subject(y_5,"rock") \wedge$
			$dc{:}description(y_5,"I\ have\ a\ rock...") \wedge$
		...	
3	200 OK	$foaf{:}Image(y_1) \wedge$	$dc{:}subject(y_1,"blues") \wedge$
			$dc{:}description(y_1,"Piano\ blues...") \wedge$
			$pic{:}color(y_1, pic{:}RedColour) \wedge$
		$foaf{:}Image(y_2) \wedge$	$dc{:}subject(y_2,"blues") \wedge$
			$dc{:}description(y_2,"Strumming\ blues...") \wedge$
			$pic{:}color(y_2, pic{:}RedColour) \wedge$
		$foaf{:}Image(y_3) \wedge$	$dc{:}subject(y_3,"blues") \wedge$
			$dc{:}description(y_3,"Blues\ jam\ art...") \wedge$
			$pic{:}color(y_3, pic{:}RedColour) \wedge$
		$foaf{:}Image(y_4) \wedge$	$dc{:}subject(y_4,"blues") \wedge$
			$dc{:}description(y_4,"piano\ man\ playing") \wedge$
			$pic{:}color(y_4, pic{:}RedColour) \wedge$
		$foaf{:}Image(y_5) \wedge$	$dc{:}subject(y_5,"blues") \wedge$
			$dc{:}description(y_5,"Blues\ brothers") \wedge$
			$pic{:}color(y_5, pic{:}RedColour) \wedge$
		...	

Table 6.9: Sample responses obtained when probing Google Images service

Term	Google	Bing	Flickr	Panoramio
`get`	✓	✓	✓	✓
`multiple`	✓	✓	✓	✓
`image`	✓	✓	✓	✓
`location-filtered`	✗	✗	✗	✓
`local`	✗	✗	✓	✓
`summarized`	✓	✓	✓	✓
`titled`	✗	✗	✓	✓
`keyword-filtered`	✓	✓	✓	✗
`tag-filtered`	✗	✗	✓	✓
`size-filtered`	✓	✓	✓	✗
`creation-filtered`	✗	✗	✓	✗
`publishing-filtered`	✗	✗	✓	✗
`license-filtered`	✗	✗	✓	✗
`colour-filtered`	✓	✓	✗	✗
`content-filtered`	✓	✓	✗	✗

Table 6.10: Mapping between features and the considered services

service can be obtained by combining the feature definitions. Additionally, when executing the service adapters, the preconditions and postconditions are checked, which allows automatic validation of the services. Finally, the service descriptions are published as Linked Data, as well as the services' output.

Some additional aspects of the approach are worth mentioning. For example, both Bing and Google Images allow filtering images by content. Bing allows image filtering by photographies, illustrations and content filtering by face or face and shoulders. Instead, Google supports 'clip art' filtering, as well as photos, illustrations and face filtering, but not face and shoulders filtering. Therefore, although both of them have the capability of filtering images by content type, the specific feature is slightly different in each of them. In order to reduce complexity, it has been assumed that both services share the `content-filtered` feature, and adapt the input appropriately through the content discovery rules.

The feature-oriented approach allows reusing features across different services, even in the case that they belong to different domains. Many of the modelled features can be combined to describe services that do not target image retrieval, but resource retrieval in general. A video search service would only need to introduce a new feature ("dealing with videos") and define a few feature definitions. Similarly, a document search service is simply a subset of the defined vocabulary (i.e. to define a web page search service, the `image` feature should be left out). Storage services (i.e. those performed with the HTTP POST and PUT methods) would require more feature definitions, as most definitions interact with `get` feature.

A metasearch service

A metasearch service has been implemented as a case study for the defined vocabulary. This metasearcher aggregates results from the described services (i.e. Google Images, Bing Images, Flickr and Panoramio). As these services have heterogeneous features, the problem is not trivial.

The metasearcher works as follows. First, it considers a service description that includes all the features shown in table 6.6. Then, by analizing the preconditions that result from combining feature definitions, it identifies required inputs and their types. With this set of inputs, it builds a user interface that allows executing the service.

However, as can be observed in table 6.6, no service has all the mentioned features. The metasearcher will then select matching services according to the features that are used in each query. In order to do so, the metasearcher validates preconditions for each feature. If the precondition is not valid, the feature is considered inactive. Only active features will be considered when filtering services. Then, the matching services are executed and their results are aggregated.

For example, let's consider that a user interacts with the metasearcher by providing some keywords, picking an image size, and clicking the submit button. Then, `location-filtered`, `tag-filtered`, `creation-filtered`, `publishing-filtered`, `license-filtered`, `coloured-filtered` and `content-filtered` features will be deactivated, as long as their preconditions are not satisfied (as long as their required inputs are not provided).

It is remarkable that the process of detecting the features to deactivate is not immediate. An insatisfied condition belongs to a feature definition which can involve many features. For example, if no keywords are passed to the metasearcher, it has to decide whether to deactivate feature `get` or feature `keyword-filtered`, as both are involved in the definition that requires keywords as input. We consider that the number of feature interactions is an indicator of the importance of a feature. Therefore, the employed criteria has been to perform the minimum number of deactivations of definitions (and thus the minimum number of deactivation of feature interactions), and an algorithm has been implemented for this purpose.

Also, the activation state of certain features cannot be identified automatically. This is the case of the `local` feature and `titled` feature, which are present only in Flickr and Panoramio, and do not set preconditions. These features are thus not considered by the metasearcher for service matching, as their inclusion would make that only Flickr and Panoramio are able to match the required features.

The metasearcher has been implemented as a generic service aggregator, in which a set of features are selected for service matching and aggregation. This means that this implementation is independent of the considered features, as the user interface is built by analyzing a microservice description, and the features are selected according to the satisfaction of their preconditions. We have also experimented on aggregating plain search services with same satisfactory results, allowing to aggregate image results as well as document results. This makes this implementation an interesting foundation for the definition of an abstract microservice aggregator based on sets of required features, which however is out of the scope of this paper.

Discussion

The feature-oriented approach for service description has been used to build a vocabulary that can be employed to build semantic service descriptions for image search services in the web. The solution is a small set of terms that illustrates the feature-oriented approach of the framework and highlights the versatility of lightweight semantics. The feature-oriented approach has provided a solution that generalizes well to other search services while it is specific to the targete domain. The vocabulary has been used to semantically describe a set of services that are aggregated on a metasearcher.

6.3.2 Services as contents

We propose a semantic service discovery process and description of existing service repositories, such as Programmable Web and Yahoo Pipes, which are two service repositories which provide plenty of services that can be reused by developers to build new web applications. The challenges behind integrating these repositories involved the problems of defining a common model, identifying relevant data and integrating and ranking the extracted data.

A metadirectory that makes use of LiMOn has been built. This metadirectory integrates heterogeneous components that can be potentially used in various web applications. More specifically, mashup applications, services, and widgets from the Web are the considered components that will be included into the metadirectory because of the repositories that have been targeted, again Programmable Web, Yahoo Pipes and Opera Widgets.

In order to make the components addressable by developers, the metadirec-

tory stores relevant metadata that can be used by the developers for selecting components. Additionally, these metadata should be available in the web in order to make it possible to automate the population of the metadirectory with real components. Usually, web component repositories usually contain metadata such as a component's name, textual description, tags or categorization. Other specific properties that depend on the nature of the component can also be found, such as inputs, endpoints, web service dependencies, or underlying formal descriptions like WSMO or WSDL.

Data harvesting and integration

In this section, we will cover how the metadirectory has been populated with components from the targeted repositories (i.e. Programmable Web, Yahoo Pipes and Opera Widgets) and how the data has been integrated.

We have defined a semantic proxy layer on top of the repositories. For each repository, we have defined the mappings between their HTML contents of their web resources and the RDF data they provide according to the model defined at section 5.3.2. To define these mappings we have used the Scraping Ontology[7] [Fernández-Villamor et al., 2010b]. This approach lets the system to have an RDF view of the unstructured data in the source repositories. With that, an automated agent crawls the source repositories and extracts the RDF data, which are then stored into the metadirectory.

Once the metadirectory is populated with components from the web, a unified categorization scheme is seeked in order to provide an homogeneous interface for querying the metadirectory. This is necessary because of the diversity that is present in the categorization of the targeted repositories. For instance, components retrieved from Programmable Web are already tagged and use their own categorization scheme. The ones from Yahoo Pipes only have the tags that have been set by the users. On the other hand, Opera Widgets repository provide components that are classified under a closed set of categories. Therefore, the components do not share a common categorization scheme, which limits the querying capabilities.

To integrate all the categorization schemes, we will define mappings between the concepts of each taxonomy. This enables querying the metadirectory by using any of the available categorization schemes without restricting the query to a particular repository. To achieve this, we will define a new categorization scheme by

[7] `http://lab.gsi.dit.upm.es/scraping.rdf`

clustering the components available in the metadirectory. This automatically built scheme will be mapped to the categorization schemes provided by Programmable Web and Opera Widgets. Additionally, a mapping between Programmable Web scheme and Opera Widgets' will be manually defined.

Automatic categorization

In this section, we will describe how to automatically build a categorization system that allows users to query the metadirectory. In many cases, components already belong to a category that was defined in their source repository. As said, both Programmable Web and Opera Widgets provide some categorization schemes, with categories such as "Tools", "Mapping", or "Sports". In the case of Yahoo Pipes repository, only tags are used to categorize each pipe.

Whenever only tags are used to categorize components, we propose the following method to build a categorization scheme based on the most common tag combinations in the component space. We will use clustering techniques to identify the most common categories in the space, and thus to define a new categorization scheme. The resulting categorization scheme will be mapped to the other schemes in section 6.3.2 to provide a uniform interface for querying the metadirectory.

To perform the clustering, components are modeled as a vector representing the tags they have:

$$a = (a_1, a_2, ..., a_n), a_i \in \{0, 1\} \tag{6.7}$$

A weighted euclidean distance between a pair of components a and b is used by the clustering algorithm:

$$d(a, b) = \sqrt{\sum_{i=1}^{n} w_i \cdot (a_i - b_i)^2} \tag{6.8}$$

The weights for each dimension are adjusted according to the popularity of the tag. This way, less relevant tags will have less weight in the measuring.

According to 6.7, an example of a simple set of components like the following:

$$\begin{aligned} foursquare &= (mapping, social, games) \\ googlemaps &= (mapping) \\ facebook &= (social) \\ bluevia &= (mapping, telephony, geolocation) \end{aligned} \tag{6.9}$$

would be represented by the next vectors:

$$\begin{aligned} foursquare &= (1,1,1,0,0) \\ googlemaps &= (1,0,0,0,0) \\ facebook &= (0,1,0,0,0) \\ bluevia &= (1,0,0,1,1) \end{aligned} \tag{6.10}$$

According to the popularity of each tag, the set of weights would be the following:

$$W = (0.375, 0.250, 0.125, 0.125, 0.125) \tag{6.11}$$

And thus some sample distances would be as follows:

$$\begin{aligned} d(bluevia, googlemaps) &= \sqrt{0.125^2 + 0.125^2} \approx 0.1768 \\ d(foursquare, facebook) &= \sqrt{0.375^2 + 0.125^2} \approx 0.3953 \\ d(facebook, bluevia) &= \sqrt{0.375^2 + 0.250^2 + 0.125^2 + 0.125^2} \approx 0.4841 \end{aligned} \tag{6.12}$$

With this we can compute the similarity between two components in the metadirectory. By using this similarity measure, we can perform some clustering to identify which are the most characteristic sets of components in the metadirectory.

A Sammon mapping has been used to represent the components and clusters [Sammon, 1969]. The Sammon's mapping function allows to perform a dimensionality reduction on the component space and map the n-dimensional space to a bidimensional one while attempting to preserve the distances between the represented vectors. This allowed us to visually estimate the number of clusters that were present in the system.

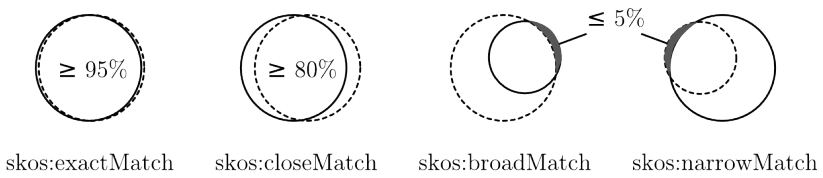

Figure 6.2: Mapping detection among categories

Mapping identification

Mappings between categorization schemes are identified automatically using an algorithm that checks set intersections. Given two categories $\mathscr{A}$ and $\mathscr{B}$ with the component sets A and B, respectively, the following mappings are identified according to the overlap between sets:

- If $\frac{|A \cap B|}{max(|A|,|B|)} \geq 0.95$, then $\mathscr{A}$ and $\mathscr{B}$ are considered equivalent categories.
- If $\frac{|A \cap B|}{max(|A|,|B|)} \geq 0.85$, then $\mathscr{A}$ and $\mathscr{B}$ are considered close categories.
- If $\frac{|A - B|}{min(|A|,|B|)} \leq 0.05$, then $\mathscr{A}$ is considered a subcategory of $\mathscr{B}$.
- If $\frac{|B - A|}{min(|A|,|B|)} \leq 0.05$, then $\mathscr{B}$ is considered a subcategory of $\mathscr{A}$.

These conditions are illustrated in Fig. 6.2. As shown, Simple Knowledge Organization System (SKOS) [Miles and Bechhofer, 2008] ontology concepts are employed to define the mappings between categories. SKOS proposes a schema for the definition of taxonomies and mappings between them. The relation $skos{:}exactMatch$ is employed for categories that are considered equivalent; $skos{:}closeMatch$ indicates that two categories are very similar and could be used interchangeably in certain contexts; $skos{:}narrowMatch$ indicates that the subject category is a subcategory of the object; $skos{:}broadMatch$ states that the subject category is a supercategory of the object.

This allowed to identify a set of mappings among the different taxonomies. Fig. 6.3 shows some of the mappings. Opera Widgets repository provides no tags, so mappings with Programmable Web's taxonomy were defined manually. As it can be seen, some categories are defined as sub- or supercategories of others, whilst others are defined as close or exact matches. In the case of Yahoo Pipes repository, the previously described method for automatically building a taxonomy was used. We executed a clustering algorithm to obtain nine different categories. Then, the

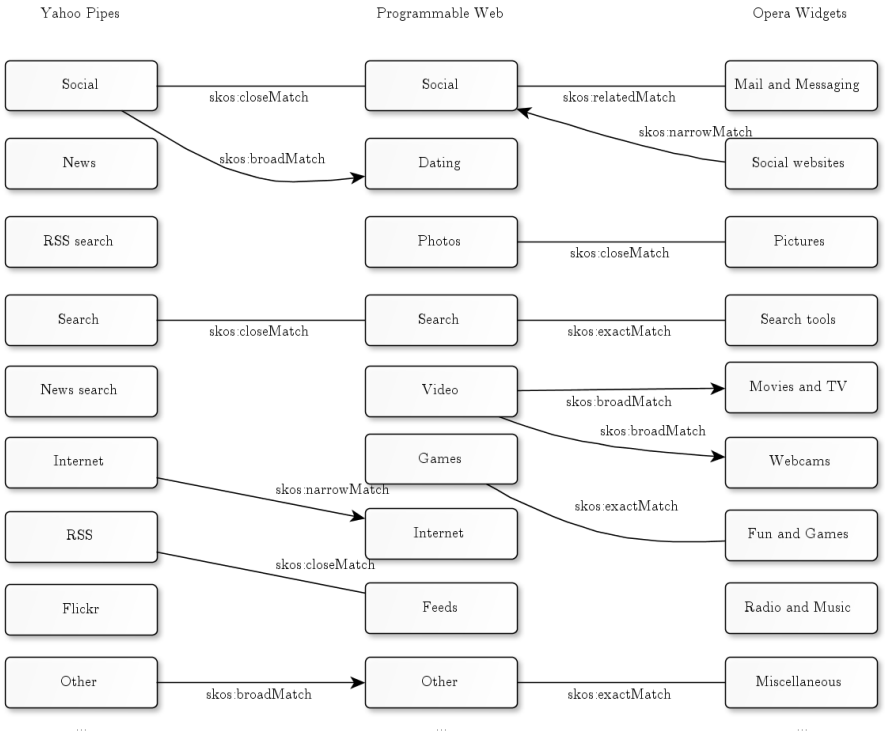

Figure 6.3: Mapping among the different categorization schemes

resulting categories were applied to Programmable Web's data. The resulting sets were compared to the ones that Programmable Web already provides as shown in this section, which resulted in the identified relations among the categories of the different taxonomies.

The metadirectory contains 10,194 services, 7,032 mashups and 1,804 widgets, as of a crawling performed in July 2011 on the mentioned repositories of Yahoo Pipes, Programmable Web and Opera Widgets.

In order to evaluate the metadirectory, a set of queries have been defined to check its selection capabilities. Table 6.11 shows the SPARQL Protocol and RDF Query Language (SPARQL) queries that result for each of the previously stated queries, along with the resulting components retrieved from the metadirectory. The queries are ones that a developer would make in order to retrieve the appropriate component for a particular problem:

1. *Which free components deal with photos/pictures?*. Usually, mashup editors offer a list of available components organized by categories. This is easily

achievable in the metadirectory by filtering components that do not match a particular category. Without a metadirectory, this would imply visiting several repositories and browsing the desired category in order to get a list of suitable components.

2. *What mapping services are provided by Microsoft?*. Because of business issues, often developers need to select components based on the provider of the component. Registries such as Programmable Web provide vendor information but do not allow filtering by vendor.

3. *Which APIs are more commonly used by telco mashups?*. Occasionally, developers need insights on component usage in a particular domain. This query is focused towards telco mashups and the kind of APIs they use. A query like this, though simple, is not allowed in the repositories where the components were extracted from.

4. *What commercial mapping services are readily usable?*. In some cases, component repositories only help to provide *awareness* of a component, i.e. know that the component exists and is available. This makes that the metadirectory is populated with components with purely general metadata such as a broad categorization and components with precise semantic description such as WSMO descriptions. I.e. some components are already runnable and others do not. Our metadirectory retrieved services from Yahoo Pipes and transformed the execution forms into WSMO service descriptions. Also, many components in Programmable Web are linked to their WSDL file. This make it possible to find readily runnable components in our metadirectory and filter them in a query.

5. *What data sources are more often employed by news mashups?*. This query insights into the sources that are employed in the field of digital news. The metadirectory allows performing the query among applications present in different repositories.

6. *Which mapping APIs are provided by the most trustable companies?*. This query attempts to select services according to their trust. We will model trust by employing provider's number of employees, as an indicator of company's size. The query could be reformulated as retrieving all APIs which belong to the mapping category, sorted by vendor's number of employees. The vendor information is retrieved from DBpedia, which

illustrates the advantage of using LiMOn for linking information on mashup components.

In comparison with searching multiple repositories manually, the metadirectory enables:

- Accessing information about components that were are originally available in separate, heterogeneous repositories from the web. As seen, component repositories offer information in their own format, which required extraction and integration in a harvesting task. After that integration, the metadirectory allows querying through a uniform interface.
- Performing complex queries about these components. Usually, component repositories such as the ones employed are very limited in their querying capabilities. Although they offer plenty of information, the offer browsing functionalities rather than complex search interfaces.
- Using external information that is available in the Linked Data cloud to complement the information from the source repositories. Information available in DBpedia or other Linked Data sources can be integrated in queries to the metadirectory, allowing to extend the queries with data that is present in other systems.

As a result, the different challenges that developers face when selecting components for building a mashup have been summarized. By using LiMOn, several component repositories have been mined and loaded onto a metadirectory. A clustering method has been used to integrate the different taxonomies of the repositories in order to unify the categorization of the metadirectory. The metadirectory then offers a unified query interface that allows retrieving components through complex queries, involving components of different nature, and allowing making use of external data from the Linked Data cloud.

6.4 Agent level

A scenario that makes use of the agent model and the discovery framework has been defined. A bare-bone implementation of the agent has been developed based on Scrappy, which was extended with the intelligent agent model. The result is an agent that is able to address top-level goals for discovering contents and services in the web and is used in a scenario to validate the proposed framework. The agent

Query	SPARQL query	Results
Which free components deal with photos/pictures?	`SELECT ?component WHERE { ?component rdf:type limon:Component; limon:categorizedBy limon:PhotoCategory; FILTER NOT EXISTS { ?component limon:usageFees ?fees . } }`	Photobucket, TweetPhoto, AOL Pictures, Lockerz, Pixlr, Moodstocks, Fonxvard, Steply, Pixenate, Fishup, Shutterfly, Picmember, ExposureManager, PicApp, and 46 more
What mapping services are provided by Microsoft?	`SELECT ?service WHERE { ?service rdf:type limon:Service; limon:categorizedBy limon:MappingCategory; limon:provider <http://www.microsoft.com> . }`	Bing Maps
Which APIs are more commonly used by telco mashups?	`SELECT (count(?api) as ?apis) ?api WHERE { ?mashup limon:uses ?api ; limon:categorizedBy limon:TelcoCategory . } GROUP BY ?api ORDER BY DESC (?apis)`	Twilio (52%), Twitter (5.7%), Tropo (3.9%), Facebook (3.6%), other (34.5%)
What commercial mapping services are readily usable?	`SELECT ?service WHERE { ?service rdf:type limon:Service; limon:categorizedBy limon:MappingCategory ; limon:usageFees ?fees ; limon:describedBy ?wsdl . }`	CDYNE IP2Geo, ArcWeb, Postcode Anywhere, PeekaCity, ShowMyIP, FraudLabs Mexico Postal Code, FraudLabs ZIPCodeWorld United States
What data sources are more often employed by news mashups?	`SELECT (count(?api) as ?apis) ?api WHERE { ?mashup limon:uses ?api ; limon:categorizedBy limon:NewsCategory ; ?api limon:categorizedBy limon:FeedCategory . } GROUP BY ?api ORDER BY DESC (?apis)`	CNN (2.18%), Google News (1.36%), NY Times (1.18%), BBC (1.09%), Yahoo News (0.91%), others (93.17%)
Which mapping APIs are provided by the most trustable companies?	`SELECT ?api ?provider ?employees WHERE { ?api limon:categorizedBy omr:mapping ; limon:provider ?provider . ?dbpcompany dbpedia-owl:wikiPageExternalLink ?provider ; dbpedia-owl:numberOfEmployees ?employees . } ORDER BY ?employees`	Nokia Ovi Maps (Nokia: 132,430 employees), Ericsson Mobile Maps (Ericsson: 90,260 employees), Bing Maps (Microsoft: 89,000 employees), Google Maps (Google: 24,400 employees), Yahoo Maps (Yahoo: 13,600 employees), others

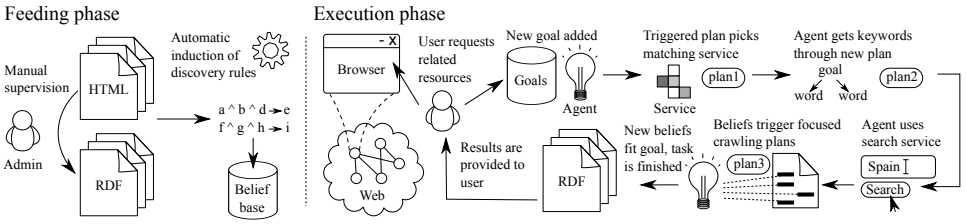

Figure 6.4: Agent's lifecycle and interaction with scenario

is then configured to perform high-level tasks under a scenario that deals with electronic newspapers.

6.4.1 Description

The scenario addresses the problem of contrasting similar pieces of news when surfing the web. Electronic newspaper readers often read a same piece of news in several sources to cross-check the views of the different newspapers, therefore spending a considerable amount of time searching the web for news. Also, users often browse news by similar topics, as they represent their own interests. Although most electronic newspapers provide recommendations to guide users when they are browsing their web sites, they do not provide outlinks to other newspapers so that the users can easily contrast different sources and how news are edited.

A contribution to the solution of this problem is using an agent that searches various electronic newspapers on behalf of the readers. For this purpose, a browser plugin is developed as a button that triggers the agent's execution. After clicking the button, the agent is launched and searches contents that might be useful for the user.

The main challenges that are addressed in this scenario are:

- Extracting the data from the addressed contents. By following the proposed discovery framework, discovery rules are used to perform this extraction. Defining the discovery rules for the information extraction is done semiautomatically thanks to rule induction algorithms.

- Using services for relating the news posts and identifying the recommendations. According to the discovery framework, the agent can use services by using feature-oriented descriptions of these services. Services will be used by the agent according to the plans that are available in the agent's plan set.

Figure 6.5: Browser plugin which shows related news by using the automated agent

As a result, two stages take place during the agent's lifecycle:

1. Feeding phase. The agent is provided with a set of HTML pages from the addressed newspapers which are annotated with the RDF data that their HTML represent. The agent then inducts discovery rules for extracting the semantic representation of the newspapers' resources. Also, service discovery rules are provided for newspapers search forms and OpenCalais [Butuc, 2009] service. OpenCalais is used to enrich the semantic descriptions of news posts to identify recommendations. OpenCalais is a service that returns Linked Data information about a piece of text, returning disambiguated entities about places or people that are mentioned in the text.

2. Execution phase. A plugin that is installed into the web browser[8] allows launching the agent with the goal of returning services that are related to the current news post that is being browsed. After clicking the button, the agent performs its focused crawling and discovers a set of results that are returned to the web browser, so that the user can review the related news

[8]In our implementation, the plugin is simply a bookmark that links to a web service which triggers the execution of the agent.

Level	Rule
Content	$uri(x,"http://abc.es") \wedge css(x,".lead") \wedge parent(x,y) \wedge$ $css(y,".headline") \Rightarrow sioc{:}Post(x) \wedge dc{:}title(x,y)$
Content	$width(x) > 70 \wedge ... \wedge parent(x,y) \wedge font\_size(y) > 12 \wedge$ $separation\_y(x,y) > 5 \wedge ... \Rightarrow sioc{:}Post(x) \wedge dc{:}title(x,y)$
Content	$css(x,"form.search") \wedge s = attr(x,"action") \Rightarrow$ $ms{:}has\_feature(s,\{ms{:}Retrieval, ms{:}KeywordFiltered,$ $ms{:}News\})$
Service	$method(s) = get \wedge status(s) = 200 \Rightarrow$ $ms{:}has\_feature(s, ms{:}Retrieval)$
Service	$x \in Input(s) \wedge y \in Output(s) \wedge ctag{:}tagged(y,x) \Rightarrow$ $ms{:}has\_feature(s, ms{:}KeywordFiltered)$
Service	$x \in Output(s) \wedge rdf{:}type(sioc{:}Post, x) \Rightarrow$ $ms{:}has\_feature(s, ms{:}News)$
Service	$x \in Input(s) \wedge rdf{:}type(sioc{:}Post, x) \wedge y \in Output(s) \wedge$ $calais{:}subject(x,y) \Rightarrow ms{:}has\_feature(s, ms{:}Related)$
Agent	$+[x, rdf{:}type, sioc{:}Post] : [z, ms{:}has\_feature, ms{:}Related] \wedge$ $[x, sioc{:}content, c] \rightarrow get(z,(body,c))$
Agent	$+[x, calais{:}subject, y] : true \rightarrow +[z, rdfs{:}label, y] \wedge$ $+![z, rdf{:}type, sioc{:}Post]$

Table 6.12: Example of rules used in the sample scenario

that the agent found in the newspapers.

These phases are shown in figure 6.4, which illustrates how the agent is managed and used.

6.4.2 Results

Table 6.12 shows the rules used to configure the agent. As said, it was configured with content rules to extract newspapers contents, with some content extraction rules automatically generated using rule induction techniques. One of them is shown in the table and uses visual features such as font size or width and height. Additionally, some feature definitions were added to the service-level knowledge base of the agent. The "related" feature is used to describe OpenCalais service, which returns related entities about a piece of news. In the case of agent-level rules,

Question	Result (1–5)
In general, are news related?	3.6 (± 0.49)
Number of newspapers is OK	3.8 (± 0.75)
Number of sections is OK	4.1 (± 0.70)
Number of posts is OK	4.2 (± 0.60)
The agent provides useful information	4.1 (± 0.70)
Is it better to surf with the agent's help?	3.8 (± 1.54)

Table 6.13: Results of users' survey

a plan for using OpenCalais service upon news retrieval was defined. A second rule is defined for using search services to retrieve news for a particular entity, where a goal for spotting a piece of news is set after an entity's label.

After the definition of discovery rules and the already present base plans, the agent was able to mine and discover contents in different newspapers. Additionally, the agent properly employed OpenCalais to enrich the semantic description of contents. After retrieving related entities for a piece of news, the agent executes the newspapers' search services with these entities to retrieve related news.

To evaluate the agent, we provided users different news posts and the recommendations by the agent in the browser plugin shown in figure 6.5. Users were then asked to answer different questions about their experience when browsing the web using the agent. Questions about the number of recommendations, the quality, the degree of relation with the original post were asked and a rating between 1 and 5 was obtained for each question.

As can be observed in table 6.13, users are overall satisfied with the agent's recommendations. This helps to validate that the agent's functionality is useful and that a system that is relevant to users has been built.

The approach has been validated on an environment of newspapers that share some visual features but which have with very different DOM tree structures. The performance of the algorithm shows high precision and good recall. After a first training stage, the system is able to extract data from sites with similar appearance, as well as keep working even on the event of changes on a web resource's DOM tree. This helps in solving the typical maintainability and generalization problems that exist in wrapper induction techniques. Thus, it is a step forward to solving the bootstrap problem of Linked Data, i.e., the lack of semantically annotated data in the Web.

6.5 Conclusions

The three levels of the framework have been evaluated in this chapter against real scenarios as part of different research projects. The content discovery level was evaluated against several projects, such as Resulta, VulneraNET, THOFU, or Contenidos a la Carta, showing the flexibility of the Scraping Ontology as well as the rule induction algorithm. The service discovery level was evaluated on ROMULUS and OMELETTE projects, where extensive service description discovery was performed in order to provide a basic set of building blocks for creating mashups. The project Contenidos a la Carta was used as well for applying agent model to a news recommender that assists users when browsing the web.

Chapter 7

Conclusions and future work

This chapter summarizes the conclusions of the thesis. As discussed in the dissertation, the current web faces a chicken and egg problem with the amount of semantically described contents and services. Although standards for semantic description of resources are available, often they are not used, which results in a lack of applications that automate the usage of semantic contents and services. To overcome this issue, the thesis proposes a unified framework for content and service discovery, and the main contributions are highlighted and discussed. Additionally, the research done has allowed to identify potential research lines.

7.1 Conclusions

One of the challenges in the current Web is the efficient use of all the available information. The Web 2.0 phenomenon has favoured the creation of contents by average users, and thus the amount of information which can be found for diverse topics has growth exponentially in the last years. Initiatives such as Linked Data attempt to build up the envisioned Semantic Web, in which a set of standards are proposed for the exchange of data among heterogeneous systems. However, these standards are sometimes not used, and still plenty of web sites require *naïve* techniques to discover the contents and services present in them.

Similarly, in the current Internet, several services are published in web applications by following SOA through either the Web Services architecture or the REST architectural style. Machine agents can automatically process service descriptions to perform execution, discovery and composition operations in an automated way. Again, the lack of semantically described services limits the applicability of automatic processing. In other words, if no services are semantically described for their automatic processing, no applications which use this services are developed, facing a chicken and egg problem that is similar to the one with semantic contents.

In this thesis, a unified framework for discovery of contents and services has been proposed in order to help the vision of the Semantic Web to gain traction and overcome the lack of semantically described resources in the web. As described in this dissertation, the discovery framework allows extracting semantic descriptions of contents and services in the web, and orchestrating the discovery thanks to an upper agent level.

The main contributions of the thesis are listed next:

- A unified discovery framework for service and content discovery. The web follows the REST architectural style, which defines a stateless hypermedia system with resources with uniform interfaces. The discovery framework is designed to suit the REST style by defining a content level for the resources representations, a service level for the actual transformations taking place in the representations, and an upper agent level for orchestration of the discovery in the basis of a human user browsing the web.

- A scraping model for content discovery. The Scraping Ontology allows to represent the mappings between the unstructured elements in a web document and the RDF resources they represent. With this ontology, it is possible to define extractors that are interoperable and are not tied

to a particular scraper or tool (e.g. PiggyBank Huynh et al. [2007]) for their interpretation and execution. Additionally, representing extractors as a transparent structure such as an RDF graph allows reasoning over the information, e.g., to extract information only from web sites that are likely to have information about flights. The mappings can be represented as rules, which result in the discovery rules used at content level.

- An algorithm for induction of content discovery rules. While content discovery rules, like other approaches like wrappers or scrapers, can be handcrafted, it is a tedious task that is overcome thanks to automated approaches such as machine learning. The algorithm for induction of discovery rules that is proposed in this thesis allows overcoming this problem while addressing issues such as robustness and generalization of rules. Thanks to the novel technique of employing visual attributes to represent, the produced content discovery rules are more resistant to changes in document layouts than other approaches and are suitable for different web sites.

- A feature-oriented approach to service description. Service description approaches consist of annotating inputs, outputs, preconditions and postconditions of services, which are modelled as functions that transform a closed universe in a particular way. This approach is formally sound but describing a service requires a big effort that is usually not taken. To reduce description efforts, a feature-oriented approach is proposed in the thesis, where services are modelled as a set of features. Features are then the reusable semantically described building blocks that are used to achieve formal service descriptions after their combination. Feature definitions can be modelled as rules, which make up the service discovery rules. This results in a lightweight description framework that reduces efforts when describing services semantically.

- An algorithm for discovery of services based on input probing. By following the discovery framework, once the contents from an HTTP interaction have been extracted, services can be discovered by analyzing the interaction and by using service discovery rules. For this task, a novel approach called service probing has been proposed in the thesis in order to achieve service discovery out of HTTP interactions. The approach is based on probing services with sets of inputs and applying service discovery rules to identify sets of features and classify services.

- A service model for service discovery. The current web has a number of registries, repositories, and API stores which offer plenty of services available for their reuse. The information available on these repositories is usually on a higher-level than typical service description frameworks, which deal with inputs, outputs and formal representations. Therefore, a higher-level model to semantically represent this information is missing. After analyzing service repositories, the LiMOn ontology has been defined in order to allow the representation of the information that is available in service repositories.

- An agent model for automation of content and service discovery. Often the problem of discovery involves crawling for a specific resource and visiting different web sites by following hyperlinks. An agent model that performs this process has been defined in the thesis. It employs the knowledge stored in content discovery rules and service discovery rules to perform a focused crawling for a specific goal. By following the BDI paradigm, an agent which follows this model can stack plans to target specific contents executing discovered services whenever appropriate.

7.2 Publications

A set of publications on conferences and journals have been produced during the development of this thesis. Most of them contain the work presented in this dissertation, although some contain works on different topics, as a result of parallel research in diverse projects. The publications are summarized next:

- José Ignacio Fernández-Villamor and Mercedes Garijo. A machine learning approach with verification of predictions and assisted supervision for a rule-based network intrusion detection system. In *Proceedings of the Fourth International Conference on Web Information Systems and Technologies*, 2008: This paper proposes an integrated Intrusion Detection System (IDS) based on induction of decision trees. It serves partly as background for the research on rule induction that is present in the thesis. [Fernández-Villamor and Garijo, 2008] is a similar paper which was published on a national conference.

- José Ignacio Fernández-Villamor, Laura Díaz-Casillas, and Carlos Á. Iglesias. A comparison model for agile web frameworks. In *Proceedings of the 2008*

Euro American Conference on Telematics and Information Systems, 2008: This paper defines a methodology for the comparison of agile web development frameworks, as part of ROMULUS project.

- José Ignacio Fernández-Villamor, Carlos Á. Iglesias, and Mercedes Garijo. Descripción semántica de aplicaciones web mediante microservicios. In *Proceedings of the Symposium on Telematics Engineering*, 2009: This paper introduces a first draft of the feature-oriented approach to service description.

- Carlos Á. Iglesias, Mercedes Garijo, José Ignacio Fernández-Villamor, and José Javier Durán Martín. Agreement Patterns. In *Workshop on Agreement Technologies (CAEPIA'09)*, pages 57–68, 2009: This paper proposes first ideas on agreement patterns which serve as background for the agent model that has been defined in the thesis.

- José Ignacio Fernández-Villamor, Carlos Á. Iglesias, and Mercedes Garijo. Microservices: lightweight service descriptions for rest architectural style. In *Proceedings of the Second International Conference on Agents and Artificial Intelligence*, 2010b: This paper defines the feature-oriented approach to service description from a methodological and theoretical view.

- José Ignacio Fernández-Villamor, Carlos Á. Iglesias, and Mercedes Garijo. A vocabulary for the modelling of image search microservices. In *Proceedings of the Fifth International Conference on Evaluation of Novel Approaches to Software Engineering*, 2010a: This paper applies the feature-oriented framework for service description to the domain of picture search services. These kinds of services were employed to provide ready-to-use services for mashup composition as part of the ROMULUS project.

- Boni García, Juan C. Dueñas, José Ignacio Fernández-Villamor, Adam Westerski, Mercedes Garijo, and Carlos A. Iglesias. Romulus: Domain driven design and mashup oriented development based on open source java metaframework for pragmatic, reliable and secure web development. In *Proceedings of the 14th European Conference on Software Maintenance and Reengineering, 2010*, Madrid, Spain, March 2010. CSMR10. URL `administrator/components/com_jresearch/files/publications/20100124_180809.pdf`: This paper summarizes the research done behind the ROMULUS project on mashup composition.

- Carlos A. Iglesias, José Ignacio Fernández-Villamor, David del Pozo, Luca Garulli, and Boni García. *Service Engineering: European research results*, chapter Combining Domain Driven Design and Mashups for Service Development, pages 171–200. Springer Verlag, 2010: This book chapter focuses on MDD by analyzing a use case and following the development methodology that was researched in the ROMULUS project.

- José Ignacio Fernández-Villamor, Jacobo Blasco-García, Carlos Á. Iglesias, and Mercedes Garijo. A semantic scraping model for web resources – applying linked data to web page screen scraping. In *Proceedings of the Third International Conference on Agents and Artificial Intelligence*, 2011: This paper introduces the Scraping Ontology by proposing an semantic proxy approach on top of the REST architectural style. It was the result of research in Contenidos a la Carta and OMELETTE projects.

- José Ignacio Fernández-Villamor, Tilo Zemke, Carlos Á. Iglesias, and Mercedes Garijo. A semantic metadirectory of services based on web mining techniques. In *Proceedings of the Association for the Advancement of Artificial Intelligence 2012 Symposia*, 2012b: This paper describes the service model for discovery that is proposed in the thesis. It summarizes the various tasks done to build the service registry that is part of the OMELETTE platform.

- Francisco Javier Blanco, José Ignacio Fernández-Villamor, and Carlos Á. Iglesias. Vulnerapedia: Security knowledge management with an ontology. In *Proceedings of the Fourth International Conference on Agents and Artificial Intelligence*, 2012: This paper describes the evaluation of the Scraping Ontology over the discovery of security contents for the construction of a unified security encyclopedia. It is part of the research done for VulneraNET project.

- Olexiy Chudnovskyy, Tobias Nestler, Martin Gaedke, Florian Daniel, José Ignacio Fernández-Villamor, Vadim Chepegin, José Ángel Fornás, Scott Wilson, Christoph Kögler, and Heng Chang. End-user-oriented telco mashups: The omelette approach. In *Proceedings of the World Wide Web Conference*, 2012: This paper summarizes the research behind the OMELETTE project, and includes research on automatic service discovery and the service model included in the thesis.

- Tilo Zemke, José Ignacio Fernández-Villamor, and Carlos Á. Iglesias. Ranking web services using centralities and social indicators. In *Proceedings of the Evaluation of Novel Approaches to Software Engineering*, 2012: This paper focuses on research on service ranking. Service ranking is a feature included in the service registry employed in the OMELETTE project.

- José Ignacio Fernández-Villamor, Carlos Á. Iglesias, and Mercedes Garijo. First-order logic rule induction for information extraction in web resources. *International Journal of Artificial Intelligence Tools*, 2012a: This paper proposes the algorithm for first-order rule induction that is used in the discovery framework to build content discovery rules automatically. It includes the evaluation on the domain of electronic newspapers, as part of the Contenidos a la Carta project. This publication is to be published in the International Journal of Artificial Intelligence (ISSN: 0218-2130, impact factor: 0.330).

7.3 Future work

The research behind this thesis has opened some research lines that deserve focus. Hence here some future work is proposed for continuing the research work done in the thesis.

Service domain classification for service probing. In the service probing algorithm, services are probed with sets of inputs that suit their preconditions. Although potentially any input can be probed, it is reasonable to probe services with inputs that provide expected behaviours so that relevant features can be discovered. A suitable approach is to automatically identify inputs for service probing out of the web resources' domain. A service domain can be identified by using bayesian classifiers on the text surrounding the execution form and the returned representation. E.g., flight search forms usually involve terms related to travel, flights, cities or airports. By classifying the domain of the service, appropriate inputs can be probed instead of monitorizing a user browsing to obtain valid interactions with services.

Content and service ranking. In addition to discovery, an agent targeting some goal usually ends up with several options or results that fit the goal. In a search or discovery problem such as the case study described in this dissertation, an immediate approach consists of showing results to the user. However, by ranking discovered resources, users can better identify the most suitable. The problem

behind this ranking function is very broad. There are many different aspects which can be used to rank contents and services, and also their degree of matching to the user goal can be taken into account. Some preliminary work on service ranking has already been undertaken in [Fernández-Villamor et al., 2012b] and [Zemke et al., 2012].

Extension of the Scraping Ontology with NLP patterns. The Scraping Ontology allows mapping elements from unstructured representations of web resources onto the RDF resources they represent. The current state of the ontology allows selecting nodes from the DOM tree and some basic selections inside the nodes' texts. In order to extract content from certain web resources, NLP patterns are a useful way to target contents otherwise impossible to extract. E.g., API documentation pages usually contain long textual explanations which include valuable information such as service parameter names, licenses, or interesting URLs. As this kind of information is embedded into texts, it belongs to the same DOM node as other non-relevant data, such as the rest of sentences that surrounds the targeted data. Similarly, occasionally it is desired to extract entities such as people names, locations or dates out of plain text. By using NLP it is possible to extract the actual information precisely. Therefore, the inclusion of NLP patterns into the Scraping Ontology is proposed as future work. A challenging aspect is the integration of NLP selectors into the algorithm for content rule induction in order to build NLP-based content discovery rules automatically.

Construction of an extended base of discovery rules. A broader library of extensive reusable feature descriptions and content types would enhance output models at each discovery level. In the thesis, a reduced subset of building blocks has been shown to solve certain use cases that involved typical elements in the web such as news posts. Further training sets for the semantic definitions would improve the generalization of the discovery rules to other scenarios that differ greatly from the ones considered, thus increasing the agent's versatility. Additionally, LiMOn can be extended with a feature orientation. The dissertation describes the model LiMOn for discovering services as contents, along with a feature-oriented approach to service description which is applied to discovery through service probing. Discovery rules that produce LiMOn-described services would leverage both approaches.

Extension of the agent model. The model can be extended to multiagent systems in order to allow cooperation between agents with different knowledges (i.e. different discovery rules in their knowledge bases). This would allow combining

and complementing discovery rules on resources. Negotiation capabilities can be added in order to allow agents to share hypothesis about web resources and select the most appropriate discovery rules and plans.

Bibliography

A9.com, inc. OpenSearch specification. `http://www.opensearch.org/Specifications/OpenSearch/1.1`, 2005.

Ben Adida and Mark Birbeck. RDFa Primer - Bridging the Human and Data Webs. `http://www.w3.org/TR/xhtml-rdfa-primer/`, 2008.

C. Alario-Hoyos and S. Wilson. Comparison of the main alternatives to the integration of external tools in different platforms. In *Proc. International Conference of Education, Research and Innovation, ICERI*, pages 3466–3476, 2010.

G. Alonso. *Web services: concepts, architectures and applications*. Springer Verlag, 2004.

E. Amoroso, T. Nguyen, J. Weiss, J. Watson, P. Lapiska, and T. Starr. Toward an approach to measuring software trust. *Published by the IEEE Computer Society*, 1991.

T. Anton. Xpath-wrapper induction by generalizing tree traversal patterns. *Lernen, Wissensentdeckung und Adaptivitt (LWA)*, pages 126–133, 2005.

S. Apel, T. Leich, and G. Saake. Aspectual mixin layers: aspects and features in concert. In *Proceedings of the 28th international conference on Software engineering*, page 131. ACM, 2006.

Sven Apel, Thomas Leich, Marko Rosenüller, and Gunter Saake. Combining feature-oriented and aspect-oriented programming to support software evolution. In *In AMSE'05, at ECOOP'05*, 2005.

Arvind Arasu. Extracting structured data from web pages. In *ACM SIGMOD*, pages 337–348, 2003.

Mark Baker. RDF Forms. http://www.markbaker.ca/2003/05/RDF-Forms/, 2005.

David T. Barnard, Gwen Clarke, and Nicolas Duncan. Tree-to-tree correction for document trees, 1995.

S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, D. McGuinness, et al. Semantic web services framework (swsf) overview. *World Wide Web Consortium, Member Submission SUBM-SWSF-20050909*, 2005.

T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

P. Bille. A survey on tree edit distance and related problems. *Theoretical computer science*, 337(1-3):217–239, 2005.

A. Birukou, E. Blanzieri, V. D'Andrea, P. Giorgini, and N. Kokash. Improving web service discovery with usage data. *IEEE software*, pages 47–54, 2007.

Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009.

M.B. Blake and M.E. Nowlan. Knowledge discovery in services (kds): Aggregating software services to discover enterprise mashups. *IEEE Transactions on Knowledge and Data Engineering*, 23(6):889–901, 2011.

Francisco Javier Blanco, José Ignacio Fernández-Villamor, and Carlos Á. Iglesias. Vulnerapedia: Security knowledge management with an ontology. In *Proceedings of the Fourth International Conference on Agents and Artificial Intelligence*, 2012.

H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.

Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. Automation and customization of rendered web pages. *Symposium on User Interface Software and Technology*, page 163, 2005. URL http://portal.acm.org/citation.cfm?id=1095062.

G. Bracha and W. Cook. Mixin-based inheritance. In *Proceedings of the European conference on object-oriented programming on Object-oriented programming systems, languages, and applications*, pages 303–311. ACM New York, NY, USA, 1990.

J.G. Breslin, S. Decker, A. Harth, and U. Bojars. SIOC: an approach to connect web-based communities. *International Journal of Web Based Communities*, 2(2): 133–142, 2006.

D. Brickley and L. Miller. Foaf vocabulary specification 0.91. Technical report, Tech. rep. ILRT Bristol, Nov. 2007. ur l: http://xmlns. com/-foaf/spec/20071002. html, 2000.

M.G. Butuc. Semantically enriching content using opencalais. *EDITIA*, 9:77–88, 2009.

Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. Extracting content structure for web pages based on visual representation. In *Proc.5 th Asia Pacific Web Conference*, pages 406–417, 2003.

J. Cao, D. Kerbyson, and G. Nudd. Use of agent-based service discovery for resource management in metacomputing environment. *Euro-Par 2001 Parallel Processing*, pages 882–886, 2001.

H. Carrillo and D. Lipman. The multiple sequence alignment problem in biology. *SIAM Journal on Applied Mathematics*, pages 1073–1082, 1988.

S. Chakrabarti, M. Van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11-16):1623–1640, 1999.

Soumen Chakrabarti and Byron Dom. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks*, 31:1623–1640, 1999.

D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin. Toward distributed service discovery in pervasive computing environments. *Mobile Computing, IEEE Transactions on*, 5(2):97–112, 2006.

C.H. Chang, M. Kayed, M.R. Girgis, and K.F. Shaalan. A survey of web information extraction systems. *IEEE Transactions on Knowledge and Data Engineering*, pages 1411–1428, 2006.

Weimin Chen. New algorithm for ordered tree-to-tree correction problem. *J. Algorithms*, 40(2):135–158, 2001. ISSN 0196-6774. doi: http://dx.doi.org/10.1006/jagm.2001.1170.

Si Won Choi and Soo Dong Kim. A Quality Model for Evaluating Reusability of Services in SOA. *Quality*, pages 293–298, 2008. doi: 10.1109/CEC/EEE.2008.55.

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

Olexiy Chudnovskyy, Tobias Nestler, Martin Gaedke, Florian Daniel, José Ignacio Fernández-Villamor, Vadim Chepegin, José Ángel Fornás, Scott Wilson, Christoph Kögler, and Heng Chang. End-user-oriented telco mashups: The omelette approach. In *Proceedings of the World Wide Web Conference*, 2012.

Philipp Cimiano, Siegfried Handschuh, Siegfried H, and Steffen Staab. Towards the self-annotating web, 2004.

Simona Colucci, Tommaso Di Noia, Eugenio Di Sciascio, M Francesco, Marina Mongiello, Giacomo Piscitelli, and Gianvito Rossi. An agency for semantic-based automatic discovery of web-services. *Discovery*, pages 1–14, 2004.

Dan Connolly. Gleaning resource descriptions from dialects of languages. `http://www.w3.org/TR/grddl/`, 2007.

Valter Crescenzi, Giansalvatore Mecca, Paolo Merialdo, UniversitÃă Roma, Tre UniversitÃă, Basilicata UniversitÃă, and Roma Tre. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of the International Conference on Very Large Databases*, pages 109–118, 2001.

F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *Internet Computing, IEEE*, 6(2):86–93, 2002.

D.A. D'Mello and VS Ananthanarayana. A review of dynamic web service description and discovery techniques. In *Integrated Intelligent Computing (ICIIC), 2010 First International Conference on*, pages 246–251. IEEE, 2010.

Pedro Domingos. Rule induction and instance-based learning: A unified approach. In *Proceedings of the 14th international joint conference on Artificial intelligence*, pages 1226–1232. Morgan Kaufmann, 1995.

Kit Eaton. Facebook won't like this apple-twitter union, 2011. URL http://www.fastcompany.com/1783467/fresh-crispy-apple-twitters-could-deep-fry-facebooks-future.

H. Elmeleegy, A. Ivan, R. Akkiraju, and R. Goodwin. Mashup advisor: A recommendation tool for mashup development. In *Web Services, 2008. ICWS'08. IEEE International Conference on*, pages 337–344. IEEE, 2008.

Tzilla Elrad, Robert E. Filman, and Atef Bader. Aspect-oriented programming: Introduction. *Commun. ACM*, 44(10):29–32, 2001. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/383845.383853.

T. Erl. *Service-oriented architecture: concepts, technology, and design*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2005.

ESSI WSMO working group. Web Service Modeling Ontology. http://www.wsmo.org/, 2004.

Oren Etzioni. Quagmire or Gold Mine? *Communications of the ACM*, 39(11): 65–68, 1996.

C. Feier and J. Domingue. WSMO primer. *DERI Working Draft, Apr*, 2005.

P. Ferguson and G. Huston. Quality of service in the internet: Fact, fiction, or compromise. *AUUGN*, page 231, 1998.

José Ignacio Fernández-Villamor and Mercedes Garijo. Sistema de detección de intrusiones con mantenimiento asistido de bases de datos de ataques mediante aprendizaje automático. In *Proceedings of the Symposium on Telematics Engineering*, 2008.

José Ignacio Fernández-Villamor and Mercedes Garijo. A machine learning approach with verification of predictions and assisted supervision for a rule-based network intrusion detection system. In *Proceedings of the Fourth International Conference on Web Information Systems and Technologies*, 2008.

José Ignacio Fernández-Villamor, Carlos Á. Iglesias, and Mercedes Garijo. A vocabulary for the modelling of image search microservices. In *Proceedings of the Fifth International Conference on Evaluation of Novel Approaches to Software Engineering*, 2010a.

José Ignacio Fernández-Villamor, Carlos Á. Iglesias, and Mercedes Garijo. Microservices: lightweight service descriptions for rest architectural style. In *Proceedings of the Second International Conference on Agents and Artificial Intelligence*, 2010b.

José Ignacio Fernández-Villamor, Jacobo Blasco-García, Carlos Á. Iglesias, and Mercedes Garijo. A semantic scraping model for web resources – applying linked data to web page screen scraping. In *Proceedings of the Third International Conference on Agents and Artificial Intelligence*, 2011.

José Ignacio Fernández-Villamor, Carlos Á. Iglesias, and Mercedes Garijo. First-order logic rule induction for information extraction in web resources. *International Journal of Artificial Intelligence Tools*, 2012a.

José Ignacio Fernández-Villamor, Tilo Zemke, Carlos Á. Iglesias, and Mercedes Garijo. A semantic metadirectory of services based on web mining techniques. In *Proceedings of the Association for the Advancement of Artificial Intelligence 2012 Symposia*, 2012b.

José Ignacio Fernández-Villamor, Laura Díaz-Casillas, and Carlos Á. Iglesias. A comparison model for agile web frameworks. In *Proceedings of the 2008 Euro American Conference on Telematics and Information Systems*, 2008.

José Ignacio Fernández-Villamor, Carlos Á. Iglesias, and Mercedes Garijo. Descripción semántica de aplicaciones web mediante microservicios. In *Proceedings of the Symposium on Telematics Engineering*, 2009.

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc 2616: Hypertext transfer protocol–http/1.1, 1999. `http://www.rfc.net/rfc2616.html`, 2009.

Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, 2000. URL `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`.

Florian Fischer and Barry Norton. D3.4.6 MicroWSMO v2 – Defining the second version of MicroWSMO as a systematic approach for rich tagging. `http://www.soa4all.eu/docs/D3.4.6+MICROWSMO_V2.PDF`, 2009.

Boni García, Juan C. Dueñas, José Ignacio Fernández-Villamor, Adam Westerski, Mercedes Garijo, and Carlos A. Iglesias. Romulus: Domain driven design

and mashup oriented development based on open source java metaframework for pragmatic, reliable and secure web development. In *Proceedings of the 14th European Conference on Software Maintenance and Reengineering, 2010*, Madrid, Spain, March 2010. CSMR10. URL `administrator/components/com_jresearch/files/publications/20100124_180809.pdf`.

Michele Gershberg. Consumers say: "in tweets we trust", 2010. URL `http://www.reuters.com/article/2010/06/23/us-retail-summit-tweets-idUSTRE65L6C320100623`.

B. Golden. *Succeeding with Open source*. Addison-Wesley Professional, 2005.

D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge Univ Pr, 1997.

Marc J. Hadley. Web application description language. `http://www.w3.org/Submission/wadl/`, 2009.

W. Harrison and H. Ossher. Subject-oriented programming: a critique of pure objects. *ACM Sigplan Notices*, 28(10):411–428, 1993.

Andrew Hogue. Thresher: Automating the unwrapping of semantic content from the world wide web. In *Proceedings of the Fourteenth International World Wide Web Conference*, pages 86–95. ACM Press, 2005.

J. Hu, C. Guo, H. Wang, and P. Zou. Quality driven web services selection. In *e-Business Engineering, 2005. ICEBE 2005. IEEE International Conference on*, pages 681–688. IEEE, 2005.

D. Huynh, S. Mazzocchi, and D. Karger. Piggy bank: Experience the semantic web inside your web browser. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(1):16–27, 2007.

Carlos Á. Iglesias, Mercedes Garijo, José Ignacio Fernández-Villamor, and José Javier Durán Martín. Agreement Patterns. In *Workshop on Agreement Technologies (CAEPIA'09)*, pages 57–68, 2009.

Carlos A. Iglesias, José Ignacio Fernández-Villamor, David del Pozo, Luca Garulli, and Boni García. *Service Engineering: European research results*, chapter Combining Domain Driven Design and Mashups for Service Development, pages 171–200. Springer Verlag, 2010.

X. Ji. Research on web service discovery based on domain ontology. In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, pages 65–68. IEEE, 2009.

Kyungkoo Jun, Krzysztof Palacz, Dan C Marinescu, and West Lafayette. Agent-Based Resource Discovery. *Sciences-New York*, 2000.

Kay Kadner and Daniel Oberle. Unified Service Description Language XG Final Report. `http://www.w3.org/2005/Incubator/usdl/XGR-usdl-20111027/`, 2011.

O. Khriyenko and M. Nagy. Semantic web-driven agent-based ecosystem for linked data and services. In *The Third International Conference on Advanced Service Computing*, pages 110–117, 2011.

M. Kirchberg, R. Kanagasabai, et al. Review of semantic web service discovery methods. In *Services (SERVICES-1), 2010 6th World Congress on*, pages 176–177. IEEE, 2010.

J. Kopeckỳ, T. Vitvar, C. Bournez, and J. Farrell. Sawsdl: Semantic annotations for wsdl and xml schema. *IEEE Internet Computing*, pages 60–67, 2007.

J. Kopecky, K. Gomadam, and T. Vitvar. hrests: An html microformat for describing restful web services. In *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, volume 1, pages 619–625. IEEE, 2008.

R. Kosala and H. Blockeel. Web mining research: A survey. *ACM SIGKDD Explorations Newsletter*, 2(1):1–15, 2000.

Nicholas Kushmerick. Wrapper induction for information extraction, 1997.

Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000.

Lea Kutvonen. Trust Aspects in the Architecture of Interoperable Systems. In *The 2nd international workshop on Interoperability solutions to Trust, Security, Policies and QoS for Enhanced Enterprise Systems*, 2007.

M. Laclavık, Z. Balogh, M. Babık, and L. Hluchỳ. Agentowl: Semantic knowledge model and agent architecture. *Computing and Informatics*, 25:419–437, 2006.

R. Lara, D. Roman, A. Polleres, and D. Fensel. A conceptual comparison of wsmo and owl-s. *Web Services*, pages 254–269, 2004.

O. Lassila and R.R. Swick. Resource description framework (RDF) model and syntax. *World Wide Web Consortium, http://www.w3.org/TR/WD-rdf-syntax*, 1999.

Kristina Lerman, Steven N. Minton, and Craig A. Knoblock. Wrapper maintenance: A machine learning approach. *Journal of Artificial Intelligence Research*, 18:2003, 2003.

Grace A Lewis and Dennis B Smith. Proceedings of the International Workshop on the Foundations of Service-Oriented Architecture (FSOA 2007). *Special report CMU/SEI-2008-SR-011*, May 2007. ISSN 0008-543X. URL `http://www.ncbi.nlm.nih.gov/pubmed/21695829`.

A. Li, X. Yang, S. Kandula, and M. Zhang. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th annual conference on Internet measurement*, pages 1–14. ACM, 2010.

R.E. Lopez-Herrejon. Understanding feature modularity in feature oriented programming and its implications to aspect oriented programming. In *ECOOP2005 PhDOOS Workshop and Doctoral Symposium, Glasgow, Scotland*, 2005.

F. Majer, M. Nussbaumer, and P. Freudenstein. Operational challenges and solutions for mashups–an experience report. In *2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), held in conjunction with 18th International World Wide Web Conference (WWW 2009)*, 2009.

D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, et al. Bringing semantics to web services: The owl-s approach. *Semantic Web Services and Web Process Composition*, pages 26–42, 2005.

N.O.A. M'Bareck and S. Tata. How to consider requester's preferences to enhance web service discovery? In *Internet and Web Applications and Services, 2007. ICIW'07. Second International Conference on*, pages 59–59. IEEE, 2007.

D.A. Menasce. Qos issues in web services. *Internet Computing, IEEE*, 6(6):72 – 75, nov/dec 2002. ISSN 1089-7801. doi: 10.1109/MIC.2002.1067740.

D.A. Menasce and V.A.F. Almeida. *Capacity Planning for Web Services: metrics, models, and methods*. Prentice Hall, 2002.

Microformats community. Microformats. `http://microformats.org/`, 2008.

A. Miles and S. Bechhofer. Skos simple knowledge organization system reference. *W3C Recommendation*, 2008.

Y. Mileva, V. Dallmeier, and A. Zeller. Mining api popularity. *Testing–Practice and Research Techniques*, pages 173–180, 2010.

M.J. Murphy, M. Dick, T. Fischer, IAO Fraunhofer, and G. Stuttgart. Towards the" Semantic Grid": A state of the art survey of Semantic Web services and their applicability to collaborative design, engineering, and procurement. *Communications of the IIMA*, 8(3):11–24, 2008.

J.D. Musa, A. Iannino, and K. Okumoto. *Software reliability: measurement, prediction, application*. McGraw-Hill, Inc., 1987.

H. Namgoong, M. Chung, K. Kim, H.S. Cho, and Y. Chung. Effective semantic web services discovery using usability. In *Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference*, volume 3, pages 5–pp. IEEE, 2006.

Azadeh Ghari Neiat, Mehran Mohsenzadeh, Rana Forsati, and Amir Masoud Rahmani. An Agent-based Semantic Web Service Discovery Framework. *2009 International Conference on Computer Modeling and Simulation*, pages 194–198, February 2009. doi: 10.1109/ICCMS.2009.75. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4797381`.

P.T. Nguyen, M.A. Babar, and J.M. Verner. Critical factors in establishing and maintaining trust in software outsourcing relationships. In *Proceedings of the 28th international conference on Software engineering*, pages 624–627. ACM, 2006.

Alberto Pan, Juan Raposo, Manuel Álvarez, Paula Montoto, Vicente Orjales, Justo Hidalgo, Lucía Ardao, Anastasio Molano, and Ángel Viña. The denodo data integration platform. *Very Large Data Bases*, page 986, 2002. URL `http://portal.acm.org/citation.cfm?id=1287369.1287456`.

M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. *The Semantic WebâĂŤISWC 2002*, pages 333–347, 2002.

Y. Park, W. Jung, B. Lee, and C. Wu. Automatic discovery of web services based on dynamic black-box testing. In *Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International*, volume 1, pages 107–114. IEEE, 2009.

Abhijit A. Patil, Swapna A. Oundhakar, Amit P. Sheth, and Kunal Verma. Meteor-s web service annotation framework. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 553–562, New York, NY, USA, 2004. ACM. ISBN 1-58113-844-X. doi: 10.1145/988672.988747. URL `http://doi.acm.org/10.1145/988672.988747`.

M. Paulk. *Capability maturity model for software*. Wiley Online Library, 1993.

C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. big'web services: making the right architectural decision. In *Proceeding of the 17th international conference on World Wide Web*, pages 805–814. ACM, 2008.

F. Canan Pembe and Tunga Güngör. A tree learning approach to web document sectional hierarchy extraction. In *Proceedings of the 2nd International Conference on Angents and Artificial Intelligence*, 2010.

M. Pirker, M. Berger, and M. Watzke. An approach for fipa agent service discovery in mobile ad hoc environments. *UbiAgents04*, `http://www.ift.ulaval.ca/~{}mellouli`, 2004.

G. Polancic, R.V. Horvat, and T. Rozman. Comparative assessment of open source software using easy accessible data. In *Information Technology Interfaces, 2004. 26th International Conference on*, pages 673 –678 Vol.1, june 2004. doi: 10.1109/ITI.2004.242703.

Geovanny Poveda-Cardona. Asistente para la creación de consultas semánticas. aplicación a la fábrica de ideas de proyectos de código abierto ubuntu ideas. Master's thesis, Universidad Politécnica de Madrid, 2011.

C. Prehofer. Feature-oriented programming: A fresh look at objects. *Lecture Notes in Computer Science*, 1241:419–443, 1997.

C. Preist. A conceptual architecture for semantic web services. *The Semantic Web–ISWC 2004*, pages 395–409, 2004.

J.R. Quinlan. *C4. 5: programs for machine learning*. Morgan Kaufmann, 1993.

R.J.R. Raj and T. Sasipraba. Web service selection based on qos constraints. In *Trends in Information Sciences & Computing*, pages 156–162. IEEE, 2010.

A. Rao. Agentspeak (l): Bdi agents speak out in a logical computable language. *Agents Breaking Away*, pages 42–55, 1996.

J. Raposo, A. Pan, M. Álvarez, and Á. Viña. Automatic wrapper maintenance for semi-structured web sources using results from previous queries. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 654–659. ACM, 2005.

O. Ratsimor, D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha. Service discovery in agent-based pervasive computing environments. *Mobile Networks and Applications*, 9(6):679–692, 2004. URL `http://www.springerlink.com/index/HP12N863244M2534.pdf`.

Eric S. Raymond. *The Cathedral and the Bazaar*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition, 1999. ISBN 1565927249.

Zia Ur Rehman, Farookh K. Hussain, and Omar K. Hussain. Towards Multi-criteria Cloud Service Selection. *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 44–48, June 2011. doi: 10.1109/IMIS.2011.99. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5976164`.

D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology, Applied Ontology. IOS Press, 2005.

J. W. Sammon. A nonlinear mapping for data structure analysis. IEEE Transactions on Computers, C-18(5):401-409, May 1969, 1969.

SAP Research. What is USDL and why do we need it. `http://www.internet-of-services.com/index.php?id=288&tx_ttnews[tt_news]=218&L=0`, 2011.

Amit P. Sheth, Karthik Gomadam, and Jon Lathem. SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. In *IEEE Computer Society*, 2007.

F. Shimba. Cloud computing: Strategies for cloud computing adoption. Dublin Institute of Technology, 2010.

N.S. Sidnal, R.S. Malashetty, and S.S. Manvi. Service discovery using software agents in semantic web. In *Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on*, pages 139–143. IEEE, 2010.

K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding Semantics to Web Services Standards. In *Proceedings of the 1st International Conference on Web Services* , 2003.

G. Spanoudakis, K. Mahbub, and A. Zisman. A platform for context aware runtime web service discovery. In *IEEE 2007 International Conference on Web Services (ICWS 2007), Salt Lake City, Utah, USA*. Citeseer, 2007.

F. Steimann. On the representation of roles in object-oriented and conceptual modelling. *Data & Knowledge Engineering*, 35(1):83–106, 2000.

K. Sycara, M. Paolucci, J. Soudry, and N. Srinivasan. Dynamic discovery and coordination of agent-based semantic web services. *Internet Computing, IEEE*, 8(3):66–73, 2004.

I. Toma and D. Foxvog. Non-functional properties in web services. *WSMO Deliverable*, 2006.

Michael Toomim, Steven M. Drucker, Mira Dontcheva, Ali Rahimi, Blake Thomson, and James A. Landay. Attaching UI enhancements to websites with end users. *Conference on Human Factors in Computing Systems*, pages 1859–1868, 2009. URL `http://portal.acm.org/citation.cfm?id=1518701.1518987`.

S. Trujillo, D. Batory, and O. Diaz. Feature oriented model driven development: A case study for portlets. In *Proceedings of the 29th international conference on Software Engineering*, pages 44–53. IEEE Computer Society, 2007.

T. Vitvar, J. Kopecky, and D. Fensel. Wsmo-lite: Lightweight semantic descriptions for services on the web. In *Proceedings of the Fifth European Conference on Web Services*, pages 77–86. Citeseer, 2007.

J. Wang, J. Zhang, P.C.K. Hung, Z. Li, J. Liu, and K. He. Leveraging fragmental semantic data to enhance services discovery. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pages 687–694. IEEE, 2011.

Y. Wang and E. Stroulia. Flexible interface matching for web-service discovery. In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, pages 147–156. IEEE, 2003.

Liu Wei, Xiaofeng Meng, and Weiyi Meng. Vision-based web data records extraction. In *WebDB*, 2006.

S. Weibel. The dublin core: a simple content description model for electronic resources. *Bulletin of the American Society for Information Science and Technology*, 24(1):9–11, 1997.

Adam Westerski and Carlos Á. Iglesias. Mining sentiments in idea management systems as a tool for rating ideas. In *Large-Scale Idea Management and Deliberation workshop. 10th International Conference on the Design of Cooperative Systems (COOP2012)*, Marseille, France, April 2012. URL `http://www.gi2mo.org/files/papers/coop2012/opinions_coop2012_paper.pdf`.

Adam Westerski, Carlos Á. Iglesias, and Fernando Tapia Rico. A model for integration and interlinking of idea management systems. In *4th Metadata and Semantics Research Conference (MTSR 2010)*, Alcalá de Henares, Spain, October 2010. URL `http://www.gi2mo.org/files/papers/mtsr/mtsr2010_gi2mo_paper.pdf`.

E. Wilde and M. Gaedke. Web Engineering Revisited. In *Proceedings of the 2008 British Computer Society (BCS) Conference on Visions of Computer Science, London, UK (September 2008)*, 2008.

Jeffrey Wong and Jason I. Hong. Making mashups with marmite: towards end-user programming for the web. *Conference on Human Factors in Computing Systems*, page 1435, 2007. URL `http://portal.acm.org/citation.cfm?id=1240842`.

World Wide Web Consortium. OWL-S: Semantic Markup for Web Services. `http://www.w3.org/Submission/OWL-S/`, 2004.

Wright State University. HTML Microformat for Describing RESTful Web Services and APIs. http://knoesis.wright.edu/research/srl/projects/hRESTs/#hRESTs, 2008.

C. Wu and E. Chang. Searching services on the web: A public web services discovery approach. In *Signal-Image Technologies and Internet-Based System, 2007. SITIS'07. Third International IEEE Conference on*, pages 321–328. IEEE, 2007.

G. Ye, C. Wu, J. Yue, and S. Cheng. A qos-aware model for web services discovery. In *Education Technology and Computer Science, 2009. ETCS'09. First International Workshop on*, volume 3, pages 740–744. IEEE, 2009.

L. Ye and B. Zhang. Discovering web services based on functional semantics. In *Services Computing, 2006. APSCC'06. IEEE Asia-Pacific Conference on*, pages 348–355. IEEE, 2006.

Jin Yu, Boualem Benatallah, Fabio Casati, and Florian Daniel. Understanding mashup development. *IEEE Internet Computing*, 12(5):44–52, 2008. ISSN 1089-7801. doi: http://dx.doi.org/10.1109/MIC.2008.114.

Tilo Zemke, José Ignacio Fernández-Villamor, and Carlos Á. Iglesias. Ranking web services using centralities and social indicators. In *Proceedings of the Evaluation of Novel Approaches to Software Engineering*, 2012.

Wenying Zeng, Yuelong Zhao, and Junwei Zeng. Cloud service and service selection algorithm research. In *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, GEC '09, pages 1045–1048, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-326-6. doi: http://doi.acm.org/10.1145/1543834.1544004. URL http://doi.acm.org/10.1145/1543834.1544004.

Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *Proceedings of the 14th international conference on World Wide Web*, pages 76–85. ACM, 2005a.

Yanhong Zhai and Bing Liu. Extracting web data using instance-based learning. In *Proc. of 6th Intl. Conf. on Web Information Systems Engineering (WISE'05*, pages 318–331, 2005b.

J. Zhang and L.-J. Zhang. Criteria analysis and validation of the reliability of web services-oriented systems. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, pages 2 vol. (xxxiii+856), july 2005. doi: 10.1109/ICWS.2005.44.

P. Zhang and J. Li. Ontology assisted web services discovery. In *Service-Oriented System Engineering, 2005. SOSE 2005. IEEE International Workshop*, pages 45–50. IEEE, 2005.

Jiehan Zhou, J.-P. Koivisto, and E. Niemela. A survey on semantic web services and a case study. In *Computer Supported Cooperative Work in Design, 2006. CSCWD '06. 10th International Conference on*, pages 1–7, May 2006. doi: 10.1109/CSCWD.2006.253254.

List of Figures

List of Tables

Glossary

Acronyms

AOP	Aspect-Oriented Programming. 43, 73
API	Application Programming Interface. xii, xiv, 12, 30, 78, 84, 85, 126, 130
BDI	Belief-Desire-Intention. 41, 126
CMM	Capability Maturity Model. 82
CSS	Content Style Sheets. xi, xiii, 48, 51–53, 58, 60, 64, 68, 95
DC	Dublin Core. 48, 57, 63, 94
DOM	Document Object Model. 7, 8, 47, 53, 54, 56–61, 63, 64, 68, 69, 95, 96, 120, 130, 149
ebXML	Electronic Business using XML. 16
FIPA	Foundation for Intelligent Physical Agents. 32
FLOWS	First-order Logic Ontology for Web Services. 23
FOAF	Friend of a Friend. 48, 94
FOMDD	Feature-Oriented Model-Driven Development. 73
FOP	Feature-Oriented Programming. 43, 73, 75
GRDDL	Gleaning Resource Descriptions from Dialects of Languages. 11

HTML	HyperText Markup Language. xi, xiii, 3, 6–8, 10, 11, 28–31, 34, 40, 42, 47–52, 55, 57–59, 61, 63, 65, 82, 95, 100, 109, 118, 149, 151
HTTP	HyperText Transfer Protocol. 4, 14, 15, 26, 28, 31, 37, 41, 43, 44, 46, 72, 73, 75–78, 90, 104, 106, 125
IOPE	Inputs Outputs Preconditions and Effects. 13, 17, 28
LiMOn	Linked Mashups Ontology. 34, 78, 79, 85, 88, 90, 108, 115, 126, 130, 149
MDD	Mashup-Driven Development. xii, xiv, 79, 92, 128
MIME	Multipurpose Internet Mail Extensions. 26, 28
NLP	Natural Language Processing. 22, 48, 54, 130
OOP	Object-Oriented Programming. 43, 73
OSSMM	Open Source Software Maturity Model. 82
OWASP	The Open Web Application Security Project. 93
OWL	Web Ontology Language. 16, 19, 20
OWL-S	OWL Services. 12, 16, 17, 19, 22, 23, 149
QoS	Quality of Service. 13, 86
RDF	Resource Description Framework. xi, xiii, 1, 10–12, 20, 29, 31, 41–43, 45, 47, 50–54, 57–59, 61–64, 68, 78, 94, 96, 109, 118, 124, 125, 130, 149
RDFa	Resource Description Framework in Attributes. 3, 29–31, 52
REST	Representational Stateless Transfer. xi–xiv, 2–4, 6, 12–15, 19, 26–29, 31–34, 37, 39–41, 44, 47, 49, 57, 73, 75, 76, 79, 92, 124, 128
ROP	Role-Oriented Programming. 73

ROSM	Resource-Oriented Service Model. 29, 31
ROWS	Rule Ontology for Web Services. 23
RSS	Really Simple Syndication. 93
SA-REST	Semantically-Annotated REST. 29–31, 34
SAWSDL	Semantic Annotations for WSDL. 12, 23
SIOC	Semantically-Interlinked Online Communities Project. 48, 53, 57, 62, 63, 94
SKOS	Simple Knowledge Organization System. 112
SLA	Service-Level Agreement. 25
SOA	Service-Oriented Architecture. 2, 79, 80, 84, 124
SOAP	Simple Object Access Protocol. 15, 28
SOP	Subject-Oriented Programming. 73
SPARQL	SPARQL Protocol and RDF Query Language. 113, 116
SWSF	Semantic Web Services Framework. 23
SWSL	Semantic Web Services Language. 23
SWSO	Semantic Web Services Ontology. 23
UDDI	Universal Description, Discovery and Integration. 14, 15
URI	Uniform Resource Identifier. 17, 19, 23, 26, 31, 43, 50–53, 59
URL	Uniform Resource Locator. 14, 15, 87, 130
USDL	Unified Service Description Language. 25
VIPS	Vision-based Page Segmentation. 10
W3C	World Wide Web Consortium. 19, 34, 78, 83, 87
WADL	Web Application Description Language. 2, 12, 13, 27–29, 31, 34
WSDL	Web Service Definition Language. 14–17, 19, 22, 23, 25, 27, 28, 33, 34, 83–85, 87, 109, 114
WSMO	Web Service Modeling Ontology. 2, 12, 13, 19, 20, 22, 23, 28, 29, 34, 78, 83, 84, 87, 109, 114

WSMX	Web Service Modelling Execution Environment. 20
WSRR	Websphere Service Registry and Repository. 16
XML	Extensible Markup Language. 11, 13, 14, 19, 26
XPath	XML Path Language. 51, 52
XSD	XML Schema Definition. 25
XSLT	Extensible Stylesheet Language Transformation. 11