Universidad Nacional del Centro de la Provincia De Buenos Aires
Facultad de Ciencias Exactas - Departamento de Computación y Sistemas
Doctorado en Ciencias De la Computación

Un Enfoque para Facilitar el Desarrollo de Sistemas Orientados a Servicios

por
Ing. Marco Crasso

Director: Prof. Dr. Marcelo Campo

Co-Director: Prof. Dr. Alejandro Zunino

Tesis presentada como requisito parcial
para optar por el título de
Doctor en Ciencias de la Computación

Tandil, Marzo de 2010

Universidad Nacional del Centro de la Provincia De Buenos Aires
Facultad de Ciencias Exactas - Departamento de Computación y Sistemas
Doctorado en Ciencias De la Computación

An Approach to Ease the Development of Service-Oriented Software

by
Eng. Marco Crasso

Advisor: Prof. Dr. Marcelo Campo

Co-Advisor: Prof. Dr. Alejandro Zunino

A Thesis report submitted in partial fulfilment
of the requirements for the degree of
Doctor in Computer Science

Tandil, March 2010

Resumen

El crecimiento de Internet, la ubicuidad de las computadoras y la disponibilidad de redes de comunicación de alta velocidad, impactaron en el diseño de los sistemas en general, reemplazando el concepto de una computadora personal aislada por el de computadoras distribuidas y conectadas. Este concepto permite a las empresas de software utilizar la Web como un medio masivo para ofrecer sus servicios. El término "servicio Web" se refiere a un componente de software que proporciona una funcionalidad específica, que puede ser consumida a través de Internet. El desarrollo de software mediante el ensamblaje de servicios independientes se conoce como paradigma de Computación Orientada a Servicios (COS).

El concepto principal detrás del paradigma COS, es la reutilización de componentes de software a través del descubrimiento, selección e integración de servicios de terceros. El desarrollo de software orientado a servicios a escala mundial, requiere que los proveedores de servicios publiquen descripciones de sus servicios Web en registros de Internet, donde los consumidores buscarán luego. El primer paso para incorporar un servicio Web en una aplicación se llama "descubrimiento" y se refiere a buscar servicios que satisfagan características requeridas. Por esta razón, la forma en que las actividades de publicación y descubrimiento son llevadas a cabo es fundamental para el desarrollo de aplicaciones que utilizan el paradigma COS.

A pesar de la importancia de las actividades de publicación y descubrimiento, todavía quedan problemas abiertos asociados con ellas, los cuales han sido reconocidos como un obstáculo que impide la adopción de este paradigma. Para hacer frente a este obstáculo, un nuevo enfoque, llamado EasySOC, para publicar y descubrir servicios Web de manera eficaz se presenta en esta tesis. Esta tesis muestra que es factible descubrir servicios Web de terceros con precisión y rapidez, si se asiste a los publicadores a construir precisas descripciones de sus servicios y a los descubridores a describir con precisión sus necesidades. Conceptualmente, el enfoque adoptado se basa en la idea de que las descripciones de servicios y las aplicaciones orientadas a servicios contienen meta-datos que representan la funcionalidad tales servicios y las necesidades de los consumidores, respectivamente. Sin embargo, generalmente estos meta-datos no son correctamente aprovechados para la facilitación del proceso de descubrimiento. Por el contrario, en esta tesis se describe cómo generar, recolectar y representar estos meta-datos convenientemente, garantizando al mismo tiempo la recuperación rápida de resultados de descubrimiento precisos.

Para validar el enfoque, se implementaron un registro de servicios y una herramienta, basada en nuevas técnicas, para ayudar a los descubridores a describir sus necesidades. Además, se emplearon descripciones de servicios Web del mundo real. Los experimentos llevados a cabo consisten en evaluaciones individuales de los componentes principales

de la implementación del enfoque EasySOC, una comparación de su eficacia de recuperación contra la de trabajos relacionados, y evaluaciones de lo que implica la generación de descripciones de servicios Web a la manera EasySOC.

Keywords: Computación Orientada a Servicios, Publicación de servicios, Descubrimiento de servicios.

Abstract

The growth of the Internet, the ubiquity of computers and the availability of high-speed communication networks, impact on system design in general, replacing the concept of an isolated personal computer with the concept of distributed and connected computers. This concept allows software companies to use the Web as a mass media to deliver their services. The term "Web Service" refers to a software component that provides a specific functionality, which can be used over Internet. Software development by assembling independent services is known as Service Oriented Computing (SOC) paradigm.

The main concept behind the SOC paradigm, is the reuse of software components through the discovery, selection and integration of third-party services. The development of service-oriented software on a global scale, requires that providers describe their Web Services and publish them in Web-based registries, in which consumers will discover them later. The first step to incorporate a Web Service into an application is called "discovery" and refers to look for services that meet required characteristics. For this reason, how publication and discovery activities are carried on is crucial for the development of applications using SOC.

Despite the importance of publication and discovery, there are still open problems associated with them, which have been recognized as an obstacle that hinders the adoption of this paradigm. To face this obstacle, a novel approach, called EasySOC, to effectively publish and discovery Web Services is presented in this thesis. This thesis shows that it is feasible to allow discoverers to accurately and promptly discover third-party Web Services by assisting publishers to populate registries with more discoverable service descriptions, and helping discoverers to precisely describe their needs. The presented approach is conceptually based on the idea that both publishers' descriptions of their services and consumers' service-oriented applications contain meta-data representing service intended functionality and consumers' discovery needs, respectively. However, usually such meta-data are neither made explicit nor exploited for the facilitation of the discovery process. Instead, this thesis describes how to generate service descriptions meta-data, and outlines how to gather and represent such meta-data in order to conveniently exploit them, while ensuring the prompt retrieval of accurate discovery results.

To validate the approach, a service registry and a tool, based on novel techniques, for assisting discoverers to describe their needs have been implemented. Besides, real-world service descriptions were employed. The conducted experiments consists of individual evaluations of the main components of EasySOC implementation, a comparison of its retrieval effectiveness against related works, and evaluations of what involves the generation of service descriptions meta-data in the EasySOC way.

Keywords: Service-Oriented Computing, Service publication, Service discovery.

Acknowledgments

After all the time of focused and dedicated work effort, I am sure that there are many people who greatly helped me, and without their support this thesis would not have been possible. Above all, I would like to thank my advisors, Marcelo Campo and Alejandro Zunino, for their constant guidance, and encouragement, and for educating by example and sharing their knowledge.

I am deeply grateful to the ISISTAN. Thanks to all the research fellows from the ISISTAN, in particular to Cristian Mateos and Juan Manuel Rodriguez, for their valuable help, suggestions and thoughts. Besides, I thank Marcelo and the rest of the crew for making the ISISTAN a place where I felt at home.

I also thank Mariano Fischer and Matías Martinez, two undergraduate students who contributed in the implementation and evaluation of the software.

I thank the National Council of Scientific and Technical Research (CONICET) for the financial support for this work.

I also thank my family members, my father, mother, brothers, uncles, and nieces. In particular, thanks to my parents, Héctor and Claudia, for helping me in some of the most challenging parts of my life. Finally, I want to thank Silvina for her unconditional love and endless support.

Marco

"Web Services have little value if other

cannot discover, access, and make sense of them"

Ian Foster in Science, May 2005.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

API Application programmatic Interface

BEEP Blocks Extensible Exchange Protocol

HTTP Hyper Text Transfer Protocol

IR Information Retrieval

NAICS North American Industry Classification System

OWL Ontology Web Language

OWL-S OWL for Services

P2P Peer to Peer

QBE Query by Example

QoS Quality of Service

RDF Resource Definition Framework

SIC Standard Industrial Classification

SMTP Simple Mail Transfer Protocol

SOAP Simple Object Access Protocol

SOC Service Oriented Computing

SQL Structured Query Language

UDDI Universal Description, Discovery and Integration

UML Unified Modeling Lanuage

UNSPSC United Nations Standard Products and Services Code

UX UDDI eXtension

VSM Vector Space Model

WSDL Web Services Description Language

WSML Web Services Modeling Language

WSMO Web Services Modeling Ontology

WWW World Wide Web

XML eXtensible Markup Language

XML-RPC XML-Remote Procedure Call

XSD XML Schema Definition

Chapter 1

Introduction

The growing complexity of software systems makes component reuse one of the most valuable tools for software engineers (Krueger, 1992). An evolutionary process currently taking place in the software industry is shifting from developing specific functionality from scratch to discovering and combining software pieces, which may be offered by third-parties (Buyya et al., 2009).

Service-oriented computing (SOC) is a new paradigm for building software systems in accordance to the aforementioned shifting (Huhns and Singh, 2005). The main goal of SOC paradigm is to support the development of distributed applications in heterogeneous environments (Erickson and Siau, 2008). SOC enables composing or assembling together distributed functionality to build software systems. This functionality comes in the form of basic building blocks called *services*. A service can be defined as a piece of functionality done by a provider who is specialized in the management of this operation. Besides, a service is wrapped with a network-addressable interface that exposes its capabilities to the outer world, while hiding implementation details that may constraint interoperability.

Indeed, structuring a large collection of modules, or services, to form a system is not a new idea (DeRemer and Kron, 1976). Clearly, the basic ideas behind SOC have been present in the software industry since a long time ago. However, the advances in distributed system technologies encourage to implement these ideas at higher levels of distribution and heterogeneity. For instance, broadband and ubiquitous connections enable to reach the Internet from everywhere and at every time, enabling a global scale marketplace of software services. In such a marketplace, providers may offer their services and consumers may contract them regardless geographical aspects, using current Web infrastructure as the communication channel. When services are implemented using standard Web languages and protocols, they are called *Web Services* (Sivashanmugam et al., 2003).

Web Service technologies have been receiving considerable attention from mayor players in the software industry. During the last decade, corporations like IBM, Microsoft and BEA have collaborated in the development of a stack of standard technologies for services that can be published, located, and invoked across the Web. Basically, this stack defines the protocols that a service provider may use for sending and receiving data, how data are serialized, how services are described, and how to publish and discover services. These efforts on the direction of standardization enable that mostly all modern programming languages support Web Services development.

Those who promote the idea of a global marketplace of Web Services, encourage providers to "describe their services using technical descriptions, service level agreements and other human consumable documents" (Erl, 2007). Moreover, such service descriptions should be made publicly available through Web-based registries. Then, providers can publish their services in a service registry establishing: the terms of engagement, technical constraints, requirements and semantic information. At the same time, service consumers can discover published services through a registry, obtain service descriptions, analyze them, and in turn select and contract services (Bichler and Lin, 2006).

This vision of the SOC paradigm shares some aspects with the concept of outsourcing in globalized business practices. During the 1980s, outsourcing became part of the business lexicon. It refers to the delegation of operations from internal production to an external entity, which is specialized in the management of these operations. Some of the common aspects between SOC and *outsourcing* impact on a software engineering process. With SOC, software companies are encouraged to focus internal resources on core competencies, delegating non-core operations to an external entity specialized in the management of these operations. This frees software factories from recruiting more developers and training them on new technologies. Typically, buying a third-party component costs 1 to 20 percent of what it would cost to develop it internally (McConnell, 2006). The process of delegating formalizes the description of the non-core operations into a contractual relationship between the consumer and the provider. Such an agreement, establishes time and costs boundaries in a distinct manner, which should prevent cost overruns. Moreover, this allows shifting among different providers of the same service functionality.

Then, at first sight one might view SOC as another software development paradigm whose strengths are modularization and reuse (Krueger, 1992) along with distribution and interoperability (Papazoglou and Heuvel, 2007). However, the main contributions of SOC are related to the entire software engineering process as follows:

- encourage to focus in-house resources on core operations,
- lower ongoing investment in internal infrastructure,
- rise of providers specialized in particular domains and competition among them,

- tighter control of budget through predictable costs, and
- high levels of flexibility to meet changing business and commercial conditions.

Research about methodologies for developing service-oriented applications produced, in last years, many approaches to connect providers and consumers of services (Papazoglou and Heuvel, 2006). A developer can describe their services and publish them in a registry, at deploy time. From the consumers' perspective, during implementation time a developer can look for services fulfilling a specific functionality, obtain their technical descriptions, select one candidate among several ones, and bind the application under development to the implementation of the selected service. In this scenario service discovery and selection is usually performed assuming that services have been properly described, thus they can be discovered, understood, and in turn accessed. Unfortunately, in an open world setting, where services built by different organizations are made available, this assumption is not necessarily true (Juan Manuel Rodriguez et al., 2010).

1.1 Motivation

Although there is a consensus about Web Service technologies and SOC promise of loose coupling between components, agility to respond to changes in requirements, transparent distributed computing and lower ongoing investments, Web Services are currently not as broadly shared and reused as expected (Wang et al., 2004). One main reason that hinders the adoption of such technologies and SOC is that efficient Web Service discovery presents many challenges (Garofalakis et al., 2006).

A good starting point to understand these challenges is the Web Services specification for implementing service registries: Universal Description, Discovery and Integration (UDDI). UDDI[1] is a standard specification sponsored by the World Wide Web Consortium (W3C), an international community where member organizations, a full-time staff, and the public work together to develop Web standards[2]. With UDDI, publishing services consists in supplying registries with providers' information and technical descriptions of the functionalities of their services in Web Service Description Language (WSDL). WSDL is an XML dialect for describing a service intended functionality using an interface with methods and arguments, in object-oriented terminology, and documentation as textual comments[3]. For example, a provider may describe the interface of a service operation as "getTemperature(zip:string):double". Regarding discovery with UDDI, a discoverer may

[1] UDDI, `http://uddi.xml.org/`
[2] W3C, `http://www.w3.org/`
[3] WSDL, `http://www.w3.org/TR/wsdl`

look for third-party services by sending keyword-based queries to a registry, which returns a list of candidate services in response. Then, the discoverer analyzes the retrieved services and selects the most suitable for his/her needs.

Unfortunately, as the number of published Web Services increases, discovering proper services using the keyword-based support provided by the UDDI standard becomes similar to finding a needle in a haystack (Garofalakis et al., 2006). Many problems related to the efficiency of UDDI-compliant approaches to service discovery may stem from the fact that current standards to describe Web Services are incorrectly or not fully employed by publishers in practice (Fan and Kambhampati, 2005). Even worse, unlike traditional software libraries, Web Service repositories rely on little meta-data to support discovery (Sabou and Pan, 2007). Therefore, if these meta-data do not properly represent the services being published, these latter will have meagre chances of being discovered.

Many approaches have arisen from recent research in the field to supply discoverers with more powerful discovery capabilities than those provided by UDDI. Mainly three research directions have been explored to tackle the problem related to Web Service discovery. Specifically, one direction proposes to enhance service descriptions instead of exploiting them as they are. The Semantic Web effort proposes to annotate Web Service descriptions using non-ambiguous concept definitions from shared ontologies (Martin et al., 2007). By assuming that services are precisely described, it is expected that finding them will be simplified at the expense of increasing development effort (Mateos et al., 2006). Unfortunately, semantics-based approaches for service discovery suffer from the typical problems associated with ontologies, namely the high complexity involved in building them (Shamsfard and Barforoush, 2004), the lack of standard ontologies, the absence of public semantically annotated Web Services (McCool, 2006), and the effort needed for adopting semantics-aware registries.

Another direction proposes Web search engines (e.g., Google) as new sources for finding Web Services (Song et al., 2007; Al-Masri and Mahmoud, 2008). As service descriptions usually reside in Web servers, the idea is to exploit the capabilities of Web search engines to crawl and index servers content. Although this approach is transparent to publishers, several studies have experimentally showed that the precision of Web search engines when looking for known services does not significantly improve, even when proper comments had been introduced in the indexed WSDL documents (Song et al., 2007). Accordingly, Web Service discoverers experience the same problems as ordinary users of Web search engines when trying to adopt this direction.

An interesting direction proposes to adapt classic Information Retrieval (IR) techniques, such as word sense disambiguation, stop-words removal, and stemming for extracting relevant information conveyed within WSDL documents. IR-based approaches gather explanatory data, in the form of keywords or terms, from Web Service descriptions and

queries for syntactically matching them during discovery. Several IR-based approaches have been evaluated with different data-sets showing that they can effectively facilitate human discoverers' tasks without requiring all the specifications of full semantic techniques (Crasso et al., 2008b). Unfortunately, such promising results cannot be generalized and may vary with different data-sets or queries, though their corresponding efforts have been rigorously evaluated.

In fact, as the underpinnings of IR-based approaches to service discovery lie in the descriptiveness of queries and service specifications, queries or WSDL documents without any representative keywords may deteriorate IR-based registries retrieval effectiveness (Juan Manuel Rodriguez et al., 2010). Moreover, a poorly written WSDL document, besides reducing its chances of being properly retrieved by a registry, hinders human discoverers' ability to understand and select the service afterward. Therefore, for such promising results to become a reality, an honest attempt from service publishers to improve their service descriptions, and from discoverers to clearly describe queries would be needed.

In spite of the intuitive implications of the use of poorly described WSDL documents against discovery, there is a lack of studies that identify, measure and provide solutions to this problem. On the other hand, there is ongoing research on measuring the cost and benefits of bad and good API design practices (Henning, 2009). However, software practitioners lack empirical evidence showing whether detected frequent API and component design practices occur in Web Services development or not, and whether any practices affect the discoverability of services. Moreover, the impact of these practices on human discoverers' ability to understand a WSDL document has been not experimentally evaluated.

Likewise, as the descriptiveness of keywords conveyed in queries is very important for service discovery , the descriptiveness of queries has recently received increasing attention from academia for its potential effects on discovery, and there are still much work to be done in this field (Crasso et al., 2009).

To sum up, the cornerstone of service-oriented development methodologies consists in: publishing, and discovering services through service registries. Apart from needing properly described services and queries, this vision requires registries capable of facing an increment on the number of published services. Therefore, it is clear the need of an approach to service publication and discovery that addresses the problems discussed above. At the same time, such an approach should take into account the lessons learned from those which have not been widely adopted because of imposing drastic changes in development practices, technologies, and infrastructure.

1.2 The Thesis

This research strives to overcome the problems associated with service discovery hindering the development of service-oriented applications. The main thesis of this research is that it is feasible to allow discoverers to efficiently outsource third-party services, without neither requiring all the specifications of full semantics-based techniques nor charging them with the task of defining highly descriptive and precise queries.

This work proposes that methods to service publication and discovery that are lighter than those based on semantics, can be a feasible way towards the materialization of service-oriented applications. To do this, this research presents EasySOC, an approach whose main goals are:

Goal 1 to assist publishers to make more discoverable services without requiring to make exhaustive annotations.

Goal 2 to allow discoverers to retrieve proper services without requiring heavy query descriptions.

Goal 3 to allow discoverers to retrieve proper services without imposing execution time overheads to the discovery process.

Central to EasySOC approach is the fact that properly described Web Services and consumers' applications, i.e., client-side applications designed to consume third-party services, may convey information for bridging the gap between services and discoverers. Specifically, the WSDL document and UDDI entry of a service may convey important information about its offered functionality, like the name and comment of each offered operation, or the category of the service, which may help to enhance its discoverability. Similarly, the source code associated with client-side applications may carry information about the functional descriptions of the potential services that can be discovered and, in turn, consumed from within those applications. This information may help to improve the descriptiveness of consumers' outsourcing intentions. Therefore, EasySOC is based on novel heuristics for gathering such pieces of information relevant to discovery, and a model for representing and retrieving gathered data.

The backbone of the EasySOC model for representing gathered data has been adapted from the Vector Space Model (VSM) (Salton et al., 1975), a classic IR model. With this model gathered keywords are represented as vectors in a multidimensional space. Then, the discovery of relevant services for a given query operates on vector comparisons. The vector standing for a query is matched onto those vectors representing published Web Services, and their spatial nearest is used for selecting candidate services. Intuitively, this requires to match a query vector against all the service vectors in order to

find nearest neighbours, which can be a compute intensive task when the number of services is large (Schmidt and Parashar, 2004). To mitigate this, the EasySOC approach adapts the VSM using a space reduction mechanism based on a document classification algorithm (Joachims, 1997). This classifier divides the vector space into sub-spaces corresponding to a pre-established set of categories. Then, a query vector is compared against the average vector of each category, in order to determine the category whose average vector maximizes vector similarity. Finally, EasySOC looks for nearest neighbors within this category only, which makes Web Service retrieval computationally efficient even with many available services.

Similarly to those approaches to Web Services discovery based on IR techniques and models, EasySOC relies on the descriptiveness of Web Service descriptions. The WSDL and UDDI specifications are well structured, which allows gathering relevant information from them effortlessly, however developers not always produce properly described Web Service descriptions. Therefore, unveiling information about a service functionality from its WSDL document can be a hard task, if not impossible. Thus, as part of this thesis a survey of common mistakes that should be avoided when creating WSDL documents is presented. Also, novel guidelines to correct the identified bad practices are introduced with this research. Therefore, following the well-known definition of anti-pattern (Brown et al., 1998), this thesis presents a novel catalog of Web Service discoverability anti-patterns.

All in all, the EasySOC approach tackles the problem of Web Service discovery, while assisting developers on making more discoverable Web Service descriptions, i.e. the approach also focuses on publication related issues.

To corroborate the approach, a registry of Web Service description and a tool for assisting discoverers to describe their needs have been implemented. The service registry implements novel techniques for gathering and processing WSDL documents, in order to bridge syntactic differences that are commonly present at this kind of documents. At the same time, the tool gathers and processes source code containing implicit information about the services needed by the discoverers. To do this, the tool uses novel techniques proposed as part of the approach.

The implementation of the approach and a publicly available data set of real world Web Service descriptions were employed for experimentation. Several experiments were conducted to assess the performance of each main part of the approach. Concretely, the classification algorithm used by the approach to reduce the search space was compared with other classifiers; the vectorial representation was compared with other ways for mapping documents onto vectors; the whole EasySOC discovery algorithm was compared with two related works; the impact of the discoverability anti-patterns on discovery and users' ability to understand WSDL documents was evaluated using three service

registries and gathering the opinions from software engineer students and practitioners. Finally, some non-functional characteristics of the implementation were assessed.

1.3 Contributions

This thesis introduces important contributions to the area of SOC, namely:

- novel heuristics to transparently gather information relevant to the discovery process from Web Service descriptions and consumers' service-oriented applications,
- a new model for representing gathered information while enabling efficient service discovery,
- a novel catalog of Web Service discoverability anti-patterns.

This work collaterally introduces additional contributions into the context of service-oriented applications development:

- a reusable set of criteria for the characterization of service discovery systems,
- a survey and a detailed analysis of Web Services discovery system alternatives,
- a novel tool for easing the development of service-oriented applications.

1.4 Organization

The rest of the document is organized as described next. Chapter 2 provides a conceptual background on the SOC paradigm, and discusses current standards and technologies materializing this paradigm.

Chapter 3 describes and analyzes the most relevant related work. It includes the description of 21 approaches to service discovery. Moreover, the chapter presents novel criteria for evaluating service registries from both functional and non-functional points of view. The results of employing such criteria on the surveyed works are also shown. The chapter ends by presenting a detailed discussing of these results and highlighting the drawbacks of current approaches.

Chapter 4 describes EasySOC, an approach to ease two essential activities of the SOC paradigm, namely publication and discovery. This chapter is divided in sections. One section is partially derived from (Crasso et al., 2008b,a) and presents the heuristics employed to gather relevant information from Web Service descriptions. Another section

is partially derived from (Crasso et al., 2009) and presents the heuristics employed to gather relevant information from consumers' applications. The section also presents the description of a model for representing gathered information, while enabling prompt and accurate service discovery. Finally, another section presents guidelines for improving the discoverability of Web Service descriptions along with a catalog of common discoverability problems and their solutions. This section is partially derived from (Juan Manuel Rodriguez et al., 2010).

Chapter 5 details the experimental evaluations that were conducted to corroborate the feasibility of the EasySOC approach.

Chapter 8 summarizes the contributions of the thesis to the area of SOC, its limitations, some perspectives for future research, and those publications that this research originated.

Chapter 2

Background

This chapter describes concepts that are essential for understanding the remainder of this thesis. In particular, section 2.1 provides an overview of the purpose and main concepts of the Service-Oriented Computing paradigm. Then, section 2.2 describes the de facto standards towards the materialization of this paradigm, namely Web Service technologies.

2.1 Service Oriented Computing

Component-based software development is a branch of software engineering that focuses on building software in which functionality is split into a number of logical software components with well-defined interfaces (Heineman and Councill, 2001). Components are designed to hide their associated implementation, to not share state, and to communicate with other components via message exchanging. Anatomically, a component can be thought as an object from the object-oriented (OO) paradigm, and the interface(s) to which the object adheres. The spirit of the component-based paradigm is that application components only know each other's interfaces, thus high levels of flexibility and reuse can be achieved (Heineman and Councill, 2001).

Service-Oriented Computing (SOC) has evolved from component-based notions to face the challenges of software development in heterogeneous distributed environments (Papazoglou and Heuvel, 2007), where interoperability is a crucial issue not yet fully addressed, nevertheless it suggests unprecedented levels of reusability. SOC is a new computing paradigm that supports the development of distributed applications in heterogeneous environments.

With SOC, developers look for independent loosely coupled software components, called *services*, to form their applications. A service-oriented application can be viewed as a

component-based application that is created by assembling two types of components: *internal*, which are those locally embedded into the application, and *external*, which are those statically or dynamically bound to a service. When building a new application, a software designer may decide to provide an implementation for some application component, or to reuse an existing implementation instead. Current literature refers to this latter as *outsourcing*. In this context, to outsource a component C means to fill the hole left by the missing functionality with the one implemented by an existing service S.

Apart from the similarities between services and a components, services are wrapped with a network-addressable interface, which exposes their capabilities to the outer world. Services may be provided by third-parties who offer specific software components as public services.

Service A service is a software component offered by a provider through a publicly available interface.

SOC is not a technology in itself; rather, it is a way of structuring or arranging other technologies to accomplish a number of other tasks (Erickson and Siau, 2008). This naturally leads to the problem of a multiplicity of definitions of SOC since many relatively similar structural arrangements of services are possible. However, the general consensus from most available definitions is that there are three starring players within the SOC paradigm: a service provider, a service consumer and a service discovery system or registry. Figure 2.1 depicts the three elements and main interactions among them.

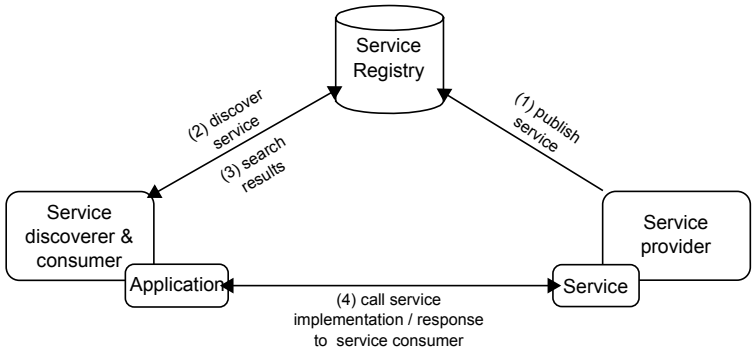

Figure 2.1: Main architecture of SOC.

Providers are entities like practitioners, companies or governments that offer services. Consumers are other entities looking for such services to integrate them into their applications. The point of the discovery system is for providers to advertise their services, so that consumers searching for such services can more easily locate and arrange to use them.

Service discovery system (a.k.a. registry) A service discovery system represents a crossroad in the path of providers and consumers. Providers can use the discovery system to advertise their services, while consumers can use it to look for services that match their needs.

The basic ideas behind SOC have been present in the software industry since a long time ago. Structuring a large collection of modules, or services, to form a system is not a new idea (DeRemer and Kron, 1976). However, SOC is pushing traditional software engineering problems, such as reuse, distribution, composition, and evolution, to their extreme. For instance, encouraged by broadband and ubiquitous connections that enable to reach the Internet from everywhere and at every time, the Web can be seen as a global scale marketplace of software services that allows providers to offer their services and consumers to contract them regardless geographical aspects. Then, at first sight one might view SOC as another software development paradigm whose strengths are modularization and reuse along with distribution and interoperability. However, the main contributions of SOC are related to the entire software engineering process.

SOC paradigm encourages focusing internal resources on core competencies, delegating non-core operations to an external entity specialized in the management of these operations. This frees software factories from recruiting more developers and training them on neither new technologies nor business domains. Typically, buying a third-party component costs 1 to 20 percent of what it would cost to develop it internally (McConnell, 2006). The process of delegating formalizes the description of the non-core operations into a contractual relationship between the consumer and the provider. Such an agreement establishes time and costs boundaries in a distinct manner, which should prevent cost overruns. This vision shares some aspects with the concept of outsourcing in globalized business practices. During the 1980s, outsourcing became part of the business lexicon. It refers to the delegation of operations from internal production to an external entity, which is specialized in the management of these operations.

Many technologies have been used for applying the SOC paradigm, including Microsoft Distributed Component Object Model (DCOM) (Brown and Kindel, 1998), Sun Remote Method Invocation (RMI) (Sun Microsystems, 1999), and Common Object Request Broker Architecture (CORBA) (Pope, 1998). These technologies were developed in the early 1990s, and while not did SOC at the time, they actually consist of the paradigm elements and activities.

The technologies mentioned in the last paragraph have been used for applying the SOC paradigm to connect intra-company applications, because they impose tight platform and protocol dependencies. Specifically, DCOM usage is essentially limited to the Windows platforms, RMI is in its beginning limited to Java applications, and CORBA uses its own stack of technologies. An evolutionary process currently taking place in the software

industry is shifting from developing specific functionality from scratch to combining third-parties functionality using technologies borrowed from the WWW. One of reason behind this fact is that service consumers want to use Web technologies (Martin-Flatin and Löwe, 2007).

2.2 Web Services

Encouraged by the growth of the Internet, the trend in distributed software development has been converging towards reusing services that can be reached using Web technologies, called *Web Services* (Vaughan-Nichols, 2002). Contrarily to DCOM, RMI and CORBA, platform neutrality and self-descriptiveness make Web Services suitable for building networks of applications distributed within and across organizational boundaries.

The term "Web Services" is used for referring to a stack of technological standards for services that can be published, located, and invoked across the Web. Then, a "Web Service" is a service that has been developed according to such standards.

Web Service A Web Service is a software component that can be discovered and invoked using standard Web protocols, while it can still be implemented in a black-box manner and invoked in a loose coupled way.

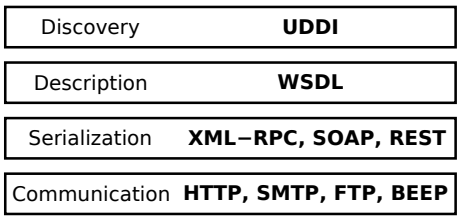

Figure 2.2: Stack of Web Service technologies (simplified).

Basically, as shown in Figure 2.2, this stack comprises four levels: (1) communication, (2) serialization, (3) description and (4) discovery. From the lowest level to the highest, the first level defines the protocols that a service provider may use for sending and receiving data. The second level is in charge of serializing the data over a communication channel. The next level defines how to describe a service, its operations and arguments. Finally, the top level specifies how to publish and discover services. In Figure 2.3 we instantiate the SOC model by using Web Services. Next subsections present the Web Service standards associated with describing and discovering services.

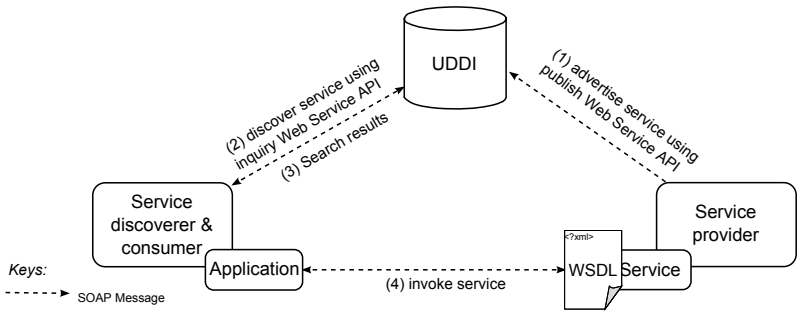

Figure 2.3: Instantiation of SOC architecture using Web Service technologies.

2.2.1 Web Service Description Language

Web Service Description Language (WSDL)[1] is an XML format for describing a service as a set of operations, whose invocation is based on message exchange. WSDL is a language that allows developers to describe two main parts of a service: its functionality and how to invoke it. Following version 1.1 of the WSDL, conceptually the functional description reveals the service interface that is offered to the outer world. The latter part specifies technological aspects, such as transport protocol and network address (a.k.a. end-points). Discoverers use the functional descriptions to match third-party services against their needs and, in turn, they take under consideration the technological details for invoking the selected service.

WSDL An XML-based language for describing the interface of a service offered to the outer world.

In object-oriented terms, the functional description of a WSDL document describes a service as an interface, an operation as a method, and a message as a method argument (inputs or outputs, indistinctly). Moreover, WSDL allows providers to describe exceptions as messages also. Optionally, each part of a WSDL document may contain documentation in the form of comments.

Technically, a WSDL document describes the service functionality as a set of *port-types*, which arrange different *operations* whose invocation is based on *message* exchange. Messages stand for the inputs or outputs of the operations, indistinctly. Exceptions are described as ordinary messages called *faults*. Besides, the main elements of a WSDL document, such as port-types, operations and their messages, must be named with unique names. Figure 2.4 depicts the Information set (Infoset) diagram of the WSDL for service interfaces.

Messages consist of *parts* that transport data between consumers and providers of services, and vice-versa. Each message part is arranged according to specific data-type

[1] WSDL, `http://www.w3.org/TR/wsdl`

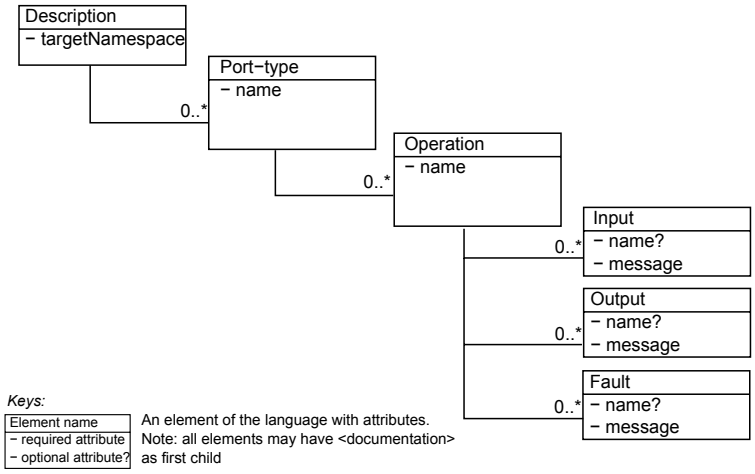

Figure 2.4: WSDL v1.1 Infoset.

definitions. The XML Schema Definition (XSD)[2] language is employed to express the structure of message parts. XSD offers constructors for defining simple types (e.g., integer and string), restrictions and both encapsulation and extension mechanisms to define more complex elements. For example, the WSDL document depicted in Figure 2.5 contains the code needed for representing a complex data-type, called "CountryCodes" which is exchanged in the input message of the "GetRate" operation. Alternatively, XSD code might be put into a separate file and imported from the WSDL document or even other WSDL documents afterward.

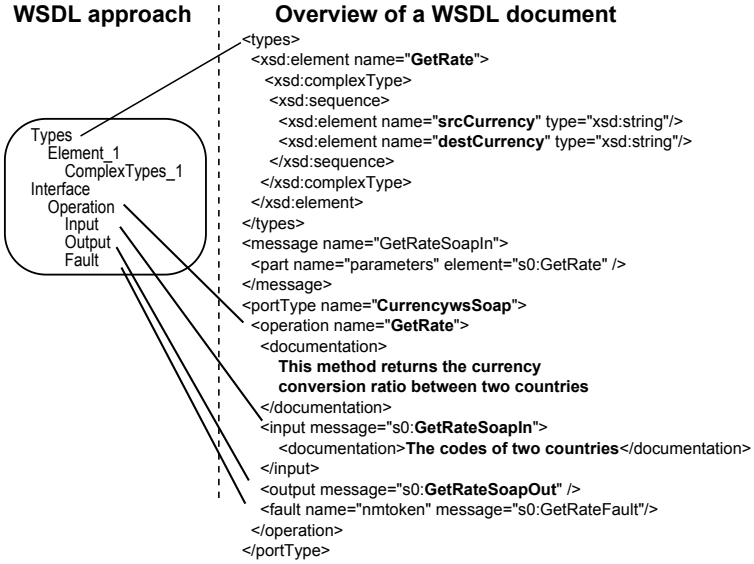

Figure 2.5: Standard Web Service description.

[2]XML Schema Part 0: Primer Second Edition, `http://www.w3.org/TR/xmlschema-0/`

WSDL allows developers to define the same message content in many encoding styles (rpc/literal, rpc/encoded, document/literal and document/literal wrapped), which may be syntactically different from each other. Broadly, a style deals with how exchanged data is encapsulated by the constructors of XSD. For example, in Figure 2.5 the argument named *currency* is defined according to the "document/literal wrapped" style. Instead, below is listed the same message content using the "document/literal" encoding style:

```
<types>
 <schema>
  <element name="srcCurrencyElement" type="xsd:string"/>
  <element name="destCurrencyElement" type="xsd:string"/>
 </schema>
</types>
...
<message name="myDocumentStyleArg">
 <part name="srcCurrency" element="srcCurrencyElement"/>
 <part name="destCurrency" element="destCurrencyElement"/>
</message>
```

To conclude the section, WSDL grammar[3] can be summarized as follows:

```
<documentation .... />?
<types>?
    <documentation .... />?
    < schema .... />*
</types>
<message name="nmtoken">*
    <documentation .... />?
    <part name="nmtoken" element="qname"? type="qname"?/>*
</message>
<portType name="nmtoken">*
    <documentation .... />?
    <operation name="nmtoken">*
        <documentation .... />?
        <input name="nmtoken"? message="qname">?
            <documentation .... />?
        </input>
        <output name="nmtoken"? message="qname">?
            <documentation .... />?
        </output>
        <fault name="nmtoken" message="qname">*
            <documentation .... />?
        </fault>
    </operation>
</portType>
```

[3]Note that "?" means optional and "*" means none or many.

2.2.2 Universal Description, Discovery and Integration

Universal Description, Discovery and Integration (UDDI)[4] is a specification of a registry for publishing and discovering Web Services. Central to UDDI purpose is the representation of meta-data about providers, in UDDI jargon "businesses", and their offered services. Apart from the data model, UDDI defines an API for advertising and discovering services. Both the model and the API are built on Web Service technologies.

UDDI A Web Services-based specification of the data model and API that a registry should manage and offer, respectively.

Conceptually, the UDDI specification arranges three groups of meta-data: White, Green and Yellow pages. White pages stand for provider/business information (e.g., address, contact, and known identifiers). Yellow pages associate a service with industrial categorizations based on standard taxonomies, such as the United Nations Standard Products and Services Code (UNSPSC)[5], the Standard Industrial Classification (SIC)[6] or the North American Industry Classification System (NAICS)[7]. Green Pages store technical information about services exposed by one or more businesses, for example a WSDL document.

In concrete terms, this specification consists of four core data-types, namely businessEntity, businessService, bindingTemplate and tModel (Technical Model), which are defined in XML. White pages are represented by businessEntity elements, Yellow pages by tModels and Green pages by businessService and bindingTemplate elements. Figure 2.6 shows the UDDI data model, associates it with a concrete example, and maps the example onto the idea of White, Green and Yellow pages.

Apart from the data model, UDDI defines one API called "Publish API" to allow service publications in a UDDI registry, and another API that provides means to look for services published in such a registry, called "Inquiry API". As UDDI is based on Web Service technologies, these APIs are defined using WSDL. The Publish API consists of 14 operations declared within a port-type. Main calls of Publish API are:

1. delete_binding: Used to remove an existing bindingTemplate from the registry.
2. delete_business: Used to delete existing businessEntity information from the registry.
3. delete_service: Used to delete an existing businessService from the registry.

[4]UDDI, `http://uddi.xml.org/`
[5]UNSPSC, `http://www.unspsc.org/`
[6]SIC, `http://www.osha.gov/pls/imis/sicsearch.html`
[7]NAICS, `http://www.naics.com`

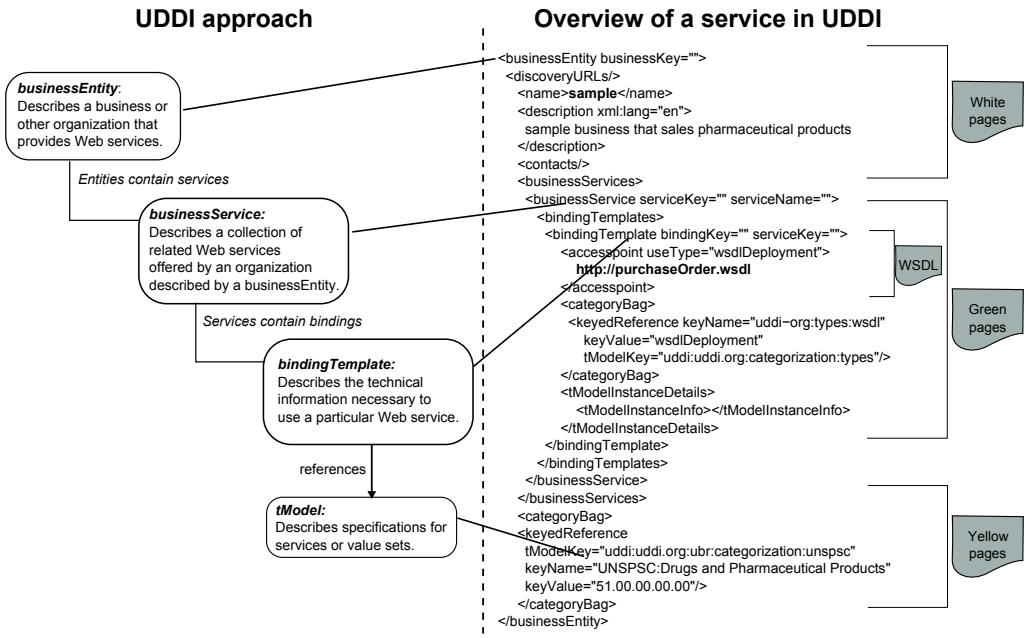

Figure 2.6: UDDI data model.

4. delete_tModel: Used to hide existing information about a tModel. Any tModel hidden in this way is still usable for reference purposes and accessible via the get_tModelDetail API, but is hidden from find_tModel result sets. There is no specified way to delete a tModel.

5. get_registeredInfo: Used to request an abbreviated list of businesses and tModels currently managed by a given publisher.

6. save_binding: Used to register new bindingTemplate information or to update existing bindingTemplate information.

7. save_business: Used to register new businessEntity information or update existing businessEntity information.

8. save_service: Used to register or update complete information about a businessService.

9. save_tModel: Used to register or update information about a tModel.

On the other hand, the Inquiry API consists of 10 operations. The main calls constituting the UDDI Inquiry API are:

1. find_binding: Used to locate bindings within or across one or more registered businessServices.

2. find_business: Used to locate information about one or more businesses.

3. find_service: Used to locate specific services within registered business entities.

4. find_tModel: Used to locate one or more tModel information structures.

From its origins UDDI has been receiving many critics (Wang et al., 2004). Some criticisms stem from the fact that UDDI model is limited to functional requirements only, instead of including non-functional properties, such as availability, cost and reliability, as well (Ran, 2003). Others claim about the poor search capabilities of an UDDI registry (Stroulia and Wang, 2005). However, UDDI advocates argue that the specification is not designed to provide powerful capabilities for discovering services, but it is designed to provide standardized formats for programmatic business and service discovery, so that search engines could be built on top of it. Therefore, UDDI represents a baseline for other approaches to Web Service discovery.

To the best of our knowledge, there are many commercial implementations of UDDI and only one open source. The Apache Software Foundation maintains jUDDI[8], the open source alternative. For an updated list of UDDI implementations, the reader should refer to the UDDI Web site[9].

2.3 Conclusions

SOC represents an emerging paradigm in distributed computing. Under the SOC model depicted in Figure 2.1, providers and consumers collaborate to build software systems. In such a model, the service discovery system, or registry, is an essential component, because it represents a crossroad in the path of providers and consumers. Providers can use the discovery system to advertise their services, while consumers can use it to discover services that match their needs.

Commonly, Web-based technologies have been widely adopted for doing SOC over the Internet. The term "Web Services" is used for referring to a stack of technology standards for modular applications that can be published, located, and invoked across the Web.

The adoption of Internet technologies encourages the idea of doing SOC on a global scale. This idea presents many requirements for a service registry. Because of the significance of service discovery for the SOC paradigm and the challenges that it presents, both researchers and industry practitioners have been developing service discovery systems. The next section describes these challenges and discusses the most relevant variety of alternatives for discovering Web Services that have been originated to face them.

[8] jUDDI, `http://ws.apache.org/juddi/`
[9] UDDI products, `http://uddi.xml.org/products`

Chapter 3

Related Work

This chapter presents works related to Web Service discovery. During the last five years, Web Services discovery has been a hot topic. Accordingly, many approaches to discover Web Services have been built. Concretely, 21 approaches to Web Service discovery have been studied in this thesis. In order to enable a fluid reading of the analyzed approaches they have been categorized in four groups: (1) those approaches that base on classic Information Retrieval techniques, (2) those that base on quality of service (QoS) information, (3) those that base on semantics, and (4) those that are highly scalable. Next subsections survey the most relevant approaches to discover Web Services according to each group. Later, eight criteria for characterizing the surveyed approaches are introduced. Concluding this chapter, survey results are discussed, putting emphasis on open problems and future research opportunities.

3.1 Information Retrieval-based approaches for discovering Web Services

Recently, some approaches have exploited classic Information Retrieval (IR) techniques, to reduce the problem of discovering relevant services to the well-known problem of finding relevant documents. Under this reduction, a "document" consists of a service description using WSDL and an entry in an UDDI registry.

Most IR-based approaches based on linear algebra have shown to be suitable alternatives for correlating similar documents. The cornerstone of such approaches is to use a vectorial representation for documents and queries and then to find out those with the most similar vectors. With the Vector Space Model (VSM) (Salton et al., 1975), a vector $\vec{v} = (e_0, ..., e_n)$ stands for a document, whose elements e_i represent the importance of each

distinct word w_i for that document. Then, similar documents will have similar vector representations.

Vector Space Model (VSM) A model for representing documents and queries as vectors, in which the likeness of vectors stands for the relevancy among the corresponding documents/queries.

Stroulia and Wang (2005); Jian and Zhaohui (2005); Platzer and Dustdar (2005); Kokash et al. (2006); Lee et al. (2007) adapt the VSM to translate Web Service descriptions into vectors. The authors also traduce keywords-based queries into vectors. Then, service look up operates by comparing vectors.

One of the main contributions of IR-based approaches is that discoverers may look up services by providing a natural language description, or a handful set of keywords. This additional capability for declaring queries, supplies discoverers with a "Google-like" inquiry interface, which most developers are familiar with. One of the main drawbacks of these approaches, on the other hand, is that they depend on publishers' use of best practices for commenting and naming services, operations and arguments. This is because the underlying matching mechanism of IR-based approaches compares strings of characters syntactically.

Other advantage of IR-based discovery approaches is their rich background inherited from previous research on classic document retrieval. In general, text mining techniques such as removing non relevant words (a.k.a. *stop-words*), bridging synonyms and removing the commoner morphological and inflectional endings from words (a.k.a. *stemming*), have been identified as being very appropriate to enhance the performance of these approaches (Korfhage, 1997). Some researchers have adapted the aforementioned techniques for dealing with syntactic differences at Web Service descriptions. For example, Dong et al. (2004); Kokash et al. (2006) remove stop-words and pull out stems from Web Service documentation. Stroulia and Wang (2005) use WordNet Lexical Database (Fellbaum, 1989) for dealing with synonyms. Other researchers have introduced new text mining techniques, for example Platzer and Dustdar (2005) bridge different WSDL message styles by mining relevant terms from data-type definitions.

Before analyzing other radically different approaches for service discovery, it is worth describing four approaches that have enhanced the classic IR-based approach with some additional techniques. Zhuge and Liu (2004) complement syntactic exact matching techniques by connecting terms that semantically include other terms, even terms that are different from a syntactical point of view. To do this, the authors propose a powerful definition of similarity between services, called flexible matching. This definition allows discoverers to look for services that are identical, more specific or more general than their queries. For example, flexible matching takes into account the distance between

two categories in a taxonomy (e.g., an available operation belongs to a category that is a subcategory of requested category), or if a requested set of keywords, which stands for operation inputs, is completely or partially met. Another contribution of this discovery approach is that developers should find its query language straightforward to learn, because it borrows the syntax and semantics from SQL (Structured Query Language). This approach uses a relational model of Web Service data based on the schema information present in UDDI.

Wang and Stroulia (2003) propose to combine a VSM based method with a structural-matching heuristic. This approach consists of two filtering stages and two kinds of queries. The first kind of query is a textual description of the expected service, which is translated into a vector and then used for retrieving relevant services from a VSM. The second stage, receives the WSDL documents most similar to the first query and compares their structure against a, possibly partial, WSDL specification of the desired service (second kind of query). The basis of this structural-matching heuristic is the comparison of the XML syntax of the data-types present in WSDL documents. Well structured documents, like WSDL ones, are formed according to a definition, i.e., an schema. A structural-matching technique deals with finding documents with similar constructions, which are valid according to the schema of a particular kind of documents.

Birukou et al. (2007) combine a VSM with information of past Web Services usage within a community of developers. The authors collect community members' query descriptions, the retrieved list of candidates for each query, and successfully invoked services as well. During discovery, this approach proposes to match new queries, requiring certain functionality, against past requests. The description of a query comprises a textual description of the discoverer's goal, the operation and its arguments. Subsequently, these descriptions are compared using a VSM, and expanding queries based on lexical relations from WordNet. Central to this approach is the idea of *implicit culture*, a concept that leads to encourage newcomers to behave similarly to more experienced members of a community.

Woogle (Dong et al., 2004) is an approach that combines multiple vector spaces with clustering techniques. Broadly, the authors propose to assess the similarity between two Web Service descriptions by separately assessing the similarity among each part of these descriptions, and then relating the individual results. To do this, they build separate vector spaces, each one for representing the documentation associated with a particular element of a WSDL document. The goal is to consider the structure of WSDL documents. Furthermore, they adapt a clustering technique for assessing the conceptual similarity between a pair of operation arguments. Clustering deals with partitioning a data-set into subsets or clusters, according to a feature that, ideally, the samples of a particular cluster share (Baeza-Yates and Ribeiro-Neto, 1999). In this sense, Woogle generates clusters for

operation parameters that share the same meaning, by assuming that a pair of parameters tends to express the same concept if both co-occur within operations (Dong et al., 2004).

Many of the aforementioned approaches have been evaluated and compared showing promising results. These evaluations assess the retrieval effectiveness of the approaches, by using classic IR measures, such as Recall, Precision and R-Precision (Korfhage, 1997).

Kokash (2006) summarizes and compares IR-based most relevant algorithms for assessing the similarity between two Web Service descriptions. The paper reports an evaluation of different algorithms using the same corpus of Web Services, concluding that algorithms based on the classic statistical measure used to evaluate how important a word is to a document, namely TF-IDF (Korfhage, 1997), over performed other approaches in most cases.

Oldham et al. (2004) show that an IR-based technique for finding similar services surpassed a technique based on graph matching, which was described in (Patil et al., 2004). Stroulia and Wang (2005) show a comparison between the two filtering stages of their previous work (Wang and Stroulia, 2003). Similarly to (Oldham et al., 2004), their results have empirically shown that "the IR method performs better than the structure-matching method" (Stroulia and Wang, 2005).

The precision of Woogle was evaluated with a public data-set of 411 Web Services, in which 25 Web Service operations represented queries (Dong et al., 2004). On the other hand, Recall was evaluated with 8 queries, i.e., operations. In terms of retrieval effectiveness, the approach has shown promising results under these test conditions –*R*-Precision was 78% and Recall was 88%–.

Birukou et al. (2007) report empirical evaluations using 20 services crawled from a public repository and 100 queries with 4 and 20 community members. These experiments have shown that the overall precision was below 80%, but precision was below 60% until, at least, 40 past evidences had been collected.

3.2 QoS-aware approaches for discovering Web Services

Although two or more services may be similar from a functional point of view, they may offer different QoS characteristics. Having only functional descriptions about services makes impossible to differentiate them by other attributes. As the UDDI model, and of those approaches based on it, is limited to discovery of functional requirements only, several approaches to find services basing on non-functional properties, such as availability, cost and reliability, have been built.

There are two approaches to incorporate QoS characteristics into the current UDDI model. In (Ran, 2003) the author proposes to augment the businessService data-type, to incor-

porate non-functional attributes for a particular service, which are organized in 5 categories, namely run-time, transaction support, configuration management, cost and security. Then, under this new model discoverers may ask for Web Services using QoS constraints. For instance, a discoverer may look for services with a desired 0.9 availability. Figure 3.1 shows a service discovery request example using Ran's model, where the attributes relevant to QoS are in bold.

```
<?xml version="1.0" encoding="UTF−8" ?>
 <envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <body>
   <find_service businessKey="*" generic="1.0"
            xmlns="urn:uddi−org:api" maxRows="100">
     <findQualifiers></findQualifiers>
     <name>Stock quote</name>
     <qualityInformation>
      <availability> 0.9 </availability>
     </qualityInformation>
   </find_service>
  </body>
 </envelope>
```

Figure 3.1: UDDI plus QoS request.

On the other hand, Overhage and Thomas (2003) revise the UDDI model and assert that it lacks of information referring to the conditions of purchase. Moreover, the authors claim that such information must be understandable for both human discoverers and software agents, which discover proper services on behalf of their users. This leads to the "UDDI Blue pages" concept (Overhage and Thomas, 2003). In this way, some semantics-aware approaches, which we will discuss in section 3.3, support non-functional descriptions of service operations.

There are different approaches to fill the aforementioned models with data that reflect actual QoS characteristics of published services. The simplest way assume that features such as response time, throughput and availability are supplied by providers. Unfortunately, that scheme raises several concerns such as integrity and reliability (Al-Masri and Mahmoud, 2007). In other words, the aforementioned features might be manipulated. To overcome this, (Makris et al., 2006) propose to recollect the duration that a service took to process a request under specific load conditions, and then deduce the performance of an immediate invocation based on gathered and current load conditions. For this purpose, every host that offers a service must implement an API for retrieving its current system state. Then, services with similar functionality are ranked according to the deduced performance and predictions based on gathered past information.

Ran (2003) affirms that a QoS-based extension must be complemented with a trusted certifier component, which verifies the claims of QoS for a Web Service before its registration. Alternatively, Zhou et al. (2004) present a different way to combine UDDI with QoS measurements, called UX (UDDI eXtension). UX instead of asking service providers

about their QoS characteristics, proposes to ask consumers. If consumers share their experiences when invoking a service, then UX will predict the future performance of a service that has been previously consumed. In this context, shared experiences are like sellers' reputation in an e-market context. Unfortunately, irresponsible consumers can lead to manipulated measures as well. Al-Masri and Mahmoud (2007) suggest that an ideal trusted broker allows publishers to provide only QoS information that cannot be inferred (e.g., cost per invocation), whereas other QoS features should be automatically computed. To do this, a broker monitors service invocations and, in turn, collects QoS information about Web Services.

Until now, different models to describe service QoS features and approaches to gather information about these features were described. Now, two approaches for exploiting collected QoS information are described. Kozlenkov et al. (2007) propose a novel query language, which allows discoverers to state QoS requirements. Basically, this work derives queries from behavioral and structural Unified Modeling Language (UML) design models of service-oriented systems. These models must be specified by using an UML extension, a.k.a. profile. This profile allows designers to indicate whether an operation must be either implemented or delegated to a third-party service. Moreover, by using this profile designers can specify desired QoS requirements about the services and operations that should be discovered. To assess the similarity between an UML-based query and available services, a two step process is used. In the first step, services with operations that satisfy the non-functional requirements of the query are retrieved. The second step uses a similarity heuristic, based on graph-matching, for finding the operations that best match the query.

Al-Masri and Mahmoud (2007) present a function for ranking Web Services according to their QoS characteristics, which had been gathered by a broker. The function computes a matrix that represents a Web Service in a row and each single QoS parameter in a column. Due to the fact that QoS parameters vary in units and magnitude, values are normalized. A second matrix is computed by dividing each column by the maximum normalized value that occurs within it. In other words, for each Web Service, any QoS parameter is divided by the maximum value for that parameter that has been collected from other services. Then, a discoverer defines a vector of weights indicating the importance level assigned to each QoS parameter. The larger the weight the more important its associated QoS parameter is to the client. The values of these weights range from 0 to 1 and all weights must add up to 1. Finally, these weights are introduced into the matrix as factors and all values are re-calculated. The row that maximizes the sum of its weighted parameters represents the first ranked Web Service, and so on.

3.3 Semantics-aware approaches for discovering Web Services

Another feature that UDDI and those approaches based on it do not support, is a machine-interpretable description of Web Services (Martin et al., 2007). Members of the Semantic Web community suggest that by annotating services with ontologies, discoverers can access to an unambiguous shared definition of each part of a Web Service (e.g., input, output, operation, etc.). For example, suppose that the output message of an operation is named "temp", then a discoverer might not be able to accurately deduce what it means. Instead, if this output is associated with a concept that defines the current temperature of a certain region, the discoverer may understand its meaning (Mateos et al., 2006).

Semantic Web Service A Web Service whose description consists of a machine interpretable definition of its constituent parts.

Moreover, ontologies support semantic matching algorithms (Euzenat and Shvaiko, 2007; Chen et al., 2006). Contrary to syntactic matching, semantic matching allows differentiating between syntactically equivalent terms, but semantically different, e.g., "temp" can stand for a temporal value or can be related to temperature. This is essential for software agents that attempt to discover services automatically, i.e., without a human discoverer's intervention. However, the promised high levels of automatism can only be achieved at the expenses of placing effort on correctly specifying ontologies, managing them and annotating services.

There are three main efforts for defining the meta model, or type of semantic information, for describing a Web Service, namely OWL-S (Martin et al., 2007), WSMO (Roman et al., 2005) and WSDL-S (Sivashanmugam et al., 2003). OWL-S (Martin et al., 2007), which was accepted as a W3C submission in November 2004, provides a framework for describing both the functions and advertisements for Web Services by using OWL (Ontology Web Language)[1]. OWL is a W3C recommendation for describing the semantic relationships of a domain. Although OWL-S includes three sub-ontologies, namely *Service Profile*, *Process Model* and *Grounding*, the *Service Profile* sub-ontology is directly related to service discovery, because it describes *what* a service does. Broadly, this sub-ontology not only allows publishers to annotate preconditions, inputs, outputs and effects of Web Service operations, but also some non-functional attributes.

OWL is designed to represent machine interpretable content on the Web. OWL is based on RDF-S, a structured language based on XML. RDF-S extends the Resource Definition Framework (RDF)[2] with a set of predefined types, high level constructors (e.g., *class*, *subClassOf*, *property*) and range and domain constraints over properties. Above RDF-S, OWL

[1] OWL, `http://www.w3.org/TR/owl-features/`
[2] RDF, `www.w3.org/RDF/`

defines more facilities (e.g., *inverseOf, equivalentClass, sameAs, symmetry*) for expressing meaning and semantics than XML, RDF, and RDF-S.

WSMO (Roman et al., 2005) is a conceptual model for Web Services, which was incorporated as a W3C submission in April 2005. It comprises *Ontologies*, *Web Services*, *Goals* and *Mediators*. The *Web Services* model allows publishers to annotate the interfaces, non-functional attributes, preconditions, post-conditions, effects and assumptions of service operations. On the other hand, the *Goals* model allows discoverers to annotate functional and non-functional requirements. In addition, the *Mediators* model can be used to define mediators responsible for aligning different ontologies. Broadly, ontology alignment, a.k.a. ontology mapping, deals with mapping one ontology onto another. This means that for each concept in one ontology, an alignment algorithm finds a corresponding entity that has the same intended meaning, but belongs to other ontology. Similarly to the OWL-S *Service Profile* ontology, these two WSMO models are relevant to discovery systems. WSMO defines these models by means of a language specially designed to express semantic descriptions according to the WSMO meta model, called Web Service Modeling Language (WSML) (de Bruijn et al., 2006). The main components of the language are concepts, attributes, binary relations and instances, as well as concept and relation hierarchies and support for data-types.

One of the main contributions of OWL-S and WSMO is the possibility of reasoning and mediating over service descriptions, which is supported by their underlying languages, i.e., OWL and WSML. Complementary to OWL-S and WSML, WSDL-S (Sivashanmugam et al., 2003) incorporates semantic descriptions into current Web Service standards. This approach uses standard extensibility elements to refer from WSDL documents to external ontologies. The semantic information specified in WSDL-S comprises definitions of the preconditions, inputs, outputs and effects of Web Service operations. Graphically, in Figure 3.2 we show this approach form a conceptual point of view along with a concrete example. This approach is loosely coupled with ontology representation languages, thus annotations in WSDL-S may be defined by using OWL, RDF, WSML or UML. WSDL-S is part of the METEOR-S project[3].

Upon the aforementioned models, different discovery systems have been developed. Paolucci et al. (2002) describe a matchmaking algorithm for Web Service descriptions written in OWL-S, which takes advantage of the underlying OWL logic to infer the logical relations between the inputs and outputs of a request, with the inputs and outputs of published semantic services. This algorithm is based on an heuristic for determining the structural similarity of a pair of concepts. Ontologically, two concepts in OWL can be equivalent (e.g., *x equivalentClass y*) or a concept can be a specialization of a superconcept (e.g., *x subClassOf y*). As a consequence, two services would be equivalent if

[3]METEOR-S Project, `http://lsdis.cs.uga.edu/projects/meteor-s/`

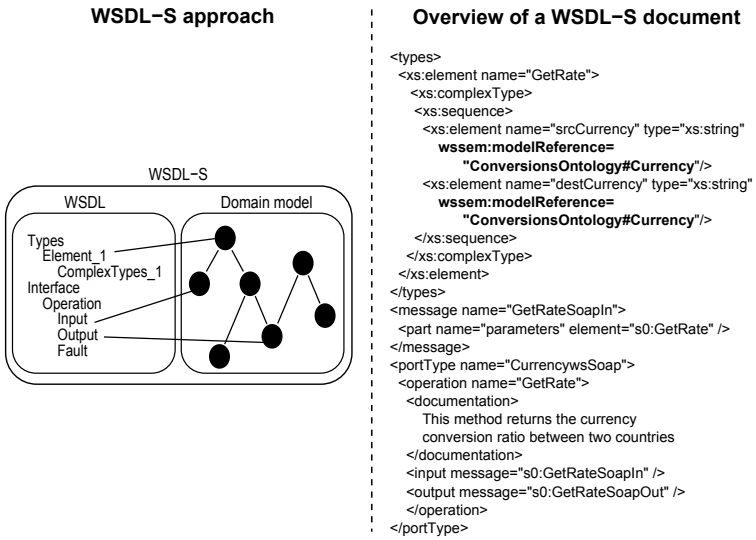

Figure 3.2: Overview of the WSDL-S approach.

their inputs and outputs are equivalent, or one would be a specialization of the another if its parameters are more specific.

Matchmaker (Kawamura et al., 2005) is a discovery system developed by Toshiba and Carnegie Mellon University, which is a further improved version of (Paolucci et al., 2002). Conceptually, this approach proposes to incorporate IR-based techniques, semantic annotations and constraint declarations into UDDI. Then, the authors propose four filtering stages for incrementally reducing the search space, namely *Namespace*, *Text*, *I/O type* and *Constraint* filter. These filters are sequentially connected. Each filter outputs 10 percent of the services received from the previous filter. The first stage of the filtering process determines whether the requested service and the registered services share, at least, one namespace. The second stage compares human-readable service documentation elements against a textual description of the desired service. At the third stage, the matching algorithm introduced in (Paolucci et al., 2002), checks whether the ontological definitions of the inputs and outputs match. Finally, the constraint filter determines whether the registered services satisfy a user's requested constraints or not.

Cardoso and Sheth (2003) also propose an algorithm to exploit syntactic and semantic information. This algorithm augments syntactic descriptions present at WSDL documents with semantic descriptions, and in turn uses both descriptions to increase matching precision. This algorithm supports Web Services described with WSMO and OWL-S, or annotated with WSDL-S. In addition, this algorithm aims at dealing with mediation between different ontologies. Here, the discovery system is part of an approach for easing work-flow instantiation based on discovering and composing Web Services.

Sivashanmugam et al. (2003) describe a semantic discovery system that follows WSDL-S for annotating and OWL for defining concepts. Broadly, the authors present a three stages algorithm that allows discoverers to search for services by specifying partially annotated services. During the first stage, a semantic matching algorithm matches Web Service operations based on the concepts that annotate their functionality. In the second stage, the result set is ranked according to the semantic similarity between input and output concepts. Optionally, the third stage ranks the result set based on the semantic similarity of precondition and effect concepts. A further improved implementation of (Sivashanmugam et al., 2003) was proposed in (Li et al., 2006).

3.4 Highly scalable and available approaches for discovering Web Services

The scalability of a service discovery systems is a crucial issue for highly distributed environments. In order to scale in the number of published services, version 3 of the UDDI specification proposes a federated environment, with a registry as a collection of one or more nodes. As such a federated architecture suffers from a single point of access and failure also (Schmidt and Parashar, 2004), the current trend consists of adapting existing discovery approaches to execute over Peer to Peer (P2P) networks (Gotthelf et al., 2008). These approaches must face the challenge of performing distributed queries to retrieve accurate results efficiently. Below the most relevant approaches in this direction are described.

Schmidt and Parashar (2004) combine a VSM-based discovery system with a P2P overlay network. Conceptually, the authors propose an indexing scheme that allows distribution of similar service descriptions on a single peer or its neighbors. In this way, a sequence of keywords that characterizes a Web Service, is mapped onto a number corresponding to the index of a physical peer. Contrary to mapping the identifier or key of a service onto the index of a peer, the cornerstone of this approach is preserving the locality of similar service descriptions, so that similar content is mapped onto the same peer. The peers of this network contain the WSDL documents of the services. As a result, this indexing scheme allows discoverers to find out which peer contains similar service descriptions given a keyword-based query.

As we explained in section 3.1, a WSDL document can be described as a sequence of n keywords representing the spatial coordinates of an n-dimensional space. Mapping a sequence of keywords onto an index, or number, is a dimension reduction problem. The implementation described in (Schmidt and Parashar, 2004), uses Hilbert's space filling curves (Sagan, 1994) for such mapping. On the other hand, it uses Chord (Stoica et al., 2003) for the overlay P2P network topology.

MWSDI (Verma et al., 2005), a part of the METEOR-S project, is an hybrid P2P-based approach for discovering Semantic Web Services. MWSDI acts as a middle-ware for inter-connecting isolated registries, in which each one offers the discovery capabilities described in (Sivashanmugam et al., 2003). The authors propose to organize distributed registries based on the domain of their published Web Services, classify a request (publication or discovery), and then redirect this request to a registry that belongs to the same domain. One or more ontologies stand for a domain, and then a Web Service registry is associated with a domain by connecting it to these ontologies, named registry ontologies. MWSDI uses a P2P approach with three types of peers. One type is in charge of managing registry ontologies. One centralized peer is responsible for coordinating new registries joining the network. Having this single point of failure makes this approach an hybrid P2P (Yang and Garcia-Molina, 2001). The third type of peer acts as the glue between a concrete registry and the infrastructure.

Sapkota et al. (2005) present another approach for combining a Semantic Web Service discovery system with a P2P. They use WSMO for describing services and matchmaking users' requests. This approach builds clusters of registries that contain similar services. A cluster consists of, at least, one super-peer, which indexes the Web Services registered within its cluster and facilitates communication between other clusters. A super-peer is dynamically selected among the members of a cluster, based on its computational capabilities and load factors. When a peer receives a discoverer's request, it attempts to resolve the query locally. When no local matching services can be found, the peer forwards the request to its super-peer. Unless the super-peer can found similar services, the request is forwarded to other clusters.

Paolucci et al. (2003) enhance their previous work (Paolucci et al., 2002) by interconnecting a collection of distributed OWL-S Matchmakers over a P2P network. Broadly, the authors propose to organize a collection of Matchmakers, broadcast a query, and then collect different results. This work proposes an architecture based on Gnutella (Steinmetz and Wehrle, 2005), an unstructured P2P approach that uses a restricted flooding method (a message has a *time to live*, which limits the number of hops before it is removed from the network) for spreading messages out to the peers.

3.5 Evaluation of discovery approaches

Previous sections described 21 approaches for enhancing UDDI capabilities to discover Web Services and Semantic Web Services. Table 3.1 highlights their benefits and issues. As the reader can see, there are essential differences among the described approaches. There are different problems related to service discovery, which range from how to describe services, to how to manage these descriptions. Since service discovery approaches

may be very different from each other, one alternative may be appropriate in a particular environment but not in others. For example, in massively distributed environments, a highly scalable solution may be a better alternative than a centralized one.

Table 3.1: Summary of the surveyed approaches to service discovery.

Main categories of evaluated approaches	Benefits	Issues
IR-based	(a) UDDI/WSDL complaint. (b) Rich background inherit from IR.	(a) Do not support for non-functional descriptions. (b) Dependency on publishers use of self-explanatory names and comments.
QoS-aware	(a) UDDI extensions. (b) Support for both functional and non-functional descriptions.	(a) Assurance of trustworthy QoS values.
Based on semantic descriptions	(a) Machine interpretable descriptions. (b) WSDL extensions. (c) Automatic discovery of services.	(a) Require ontologies. (b) Require to manage shared and distributed ontologies. (c) Require to annotate services. (d) Providers' trustworthiness assurance.
Highly scalable and available	(a) Support for a highly distributed number of providers and clients.	(a) The more the services, the more efficient the underlying discovery mechanism should be. (b) Querying distributed databases.

There is a clear need for a criteria to characterize approaches to service discovery. Toma et al. (2005) present a characterization of service discovery systems along an analysis of five alternatives. Afterward, the authors propose a combination of the most relevant features from different approaches as aspects that should be considered when developing a discovery mechanism. Garofalakis et al. (2006) present a comprehensive survey of methods, architectures and models related to Web Service discovery. The authors have thoroughly analyzed over 30 approaches for discovering Web Services (including industrial and academical ones). They have developed a taxonomy for organizing these approaches, based on four main categories: Architecture, Standards, Data model and QoS-aware.

Below, eight novel criteria are employed for characterizing Web Service discovery systems, which cover both functional and non-functional aspects. These criteria have been designed taking into account the requirements for discovering services within common service-oriented environments, in order to allow designers of service-Oriented systems

Table 3.2: Summary of characterization criteria.

Criterion group	Criterion name	Accepted Values	Mutually exclusive
Functional	Recoverable information	business, functional, non-functional, meta-model	no
	Query language	keyword, natural language, constraint, template, meta-model	no
	Mediation support	syntactic mediation, semantic matching, ontology alignment, none	yes
	Results granularity	service, operation	no
Non-functional	Scalability	a growth in the number of "x"	no
	Availability	centralized, federated, hybrid, P2P	yes
	Provider abstraction	brokering, matchmaking	yes
	Standards conformance	name of the adopted standard/s, none	no

to analyze whether an evaluated registry fits these requirements or not. In opposition to previous related works, accompanying each criterion definition a fixed range of values that the criterion accepts is presented. Table 3.2 summarizes these criteria. The criteria are divided in two groups. The first group is intended to evaluate functional features of discovery systems. The second group focuses on non-functional characteristics of discovery systems. Accompanying the presentation of each group of criteria, the results of using them to evaluate the 21 analyzed discovery approaches are shown.

3.5.1 Scalability

This non-functional criterion evaluates whether a system is designed for dealing with a growth of the number of published services, categories, nodes or not. Accepted values are: *number of nodes*, *number of services*, *number of categories* and *none*. A "number of x" value tells that an approach has been designed for managing a growth in x. In this context, the term "node" is used according to the UDDI specification, and the terms "categories" and "domains" as being indistinct. In addition, these values are not exclusive.

Scalability determines for which environments a discovery systems is appropriate. Achieving high levels of scalability is more appropriate for environments where the number of services changes frequently and there are no clear boundaries on the expected scalability. For instance, P2P systems are more appropriate in dynamic environments, e.g., in ubiquitous computing where nodes leave or join a network in unpredictable manners. However, an scalable infrastructure may be challenging from several points of views (manageability, uncertainty, deployability, etc.) (Gotthelf et al., 2008). Instead, if the number of services is known beforehand, a non-scalable infrastructure may be a good choice.

Centralized registries are more appropriate when service publications occur sporadically.

3.5.2 Availability

This non-functional criterion evaluates how robust a system is. The possible values this criterion can take are: *centralized*, *federated*, *hybrid* or *P2P*. From an architectural point of view, the first value refers to centralized approaches with a single point of failure. The *federated* value refers to a collection of distributed nodes governed by a master registry, which, in fact, acts as a single point of access and failure. On the contrary, the *P2P* value focuses on decentralized approaches, like distributed connected peers. The *hybrid* value refers to an architectural approach that is between the aforementioned two, for example a P2P network with a centralized super-peer, but with mechanisms for recovering when the super-peer fails. Similarly to the scalability criterion, achieving respectable levels of availability with centralized approaches may be rather hard. In addition, availability may even be not crucial in scenarios where discovery occurs sporadically.

3.5.3 Provider abstraction

This non-functional criterion deals with differentiating discovery systems that interfere in the interaction between requested and matched services, from those that merely match requested and published services. The valid values for this criterion are: *brokering* and *matchmaking*.

This criterion has several implications related to the availability criterion. A *brokering* approach allows better decoupling of consumers and services. A broker governs both discovery and invocation, because the broker is responsible for locating an appropriate service, forwarding the request to its provider and transmitting results and exceptions back to the service consumer. This approach imposes stronger availability requirements than matchmaking systems. Instead, when a *matchmaking* approach is used, once a service has been discovered, it may be consumed even if the discovery system is unavailable.

3.5.4 Standards conformance

This non-functional criterion evaluates whether a discovery system follows standards. The accepted values for this criterion are the names of the standards that have been adopted by the system under review. For example, accepted non-exclusive values are: *UDDI*, *WSDL*, *WSDL-S*, *OWL-S* and *WSMO*. In general, these standards are well-founded and documented. One of the main implications of following well-founded and documented practices, is that complaint discovery systems may be potentially easier to implement and adopt than ad-hoc solutions.

Tables 3.3 and 3.4 show the results, in terms of non-functional aspects, after characterizing the 21 described approaches. Those approaches that adhere to WSDL-S also adhere to WSDL, because the WSDL-S specification extends WSDL by means of WSDL supported extensibility elements, thus a WSDL-S document is also a valid WSDL document. The approaches that do not explicit adhere to any standard have been omitted from the table. For instance, the approach described in (Birukou et al., 2007) matches a request against past requests made by experienced community members. However, despite it can be theoretically integrated with UDDI registries or any other IR-based approaches, it does not explicitly adopt any standard. It is worth noting that the query language presented in (Kozlenkov et al., 2007) follows UML.

Table 3.3: Non-functional characterization of Web Service discovery systems.

Evaluated approach	Scalability	Availability	Provider abstraction
(Sivashanmugam et al., 2003; Ran, 2003; Zhou et al., 2004; Zhuge and Liu, 2004)	number of nodes	federated	matchmaking
(Al-Masri and Mahmoud, 2007)	number of nodes	federated	brokering
(Wang and Stroulia, 2003; Cardoso and Sheth, 2003; Dong et al., 2004; Platzer and Dustdar, 2005; Jian and Zhaohui, 2005; Kawamura et al., 2005; Makris et al., 2006; Kokash et al., 2006; Lee et al., 2007; Kozlenkov et al., 2007)	none	centralized	matchmaking
(Schmidt and Parashar, 2004)	number of services	P2P	matchmaking
(Verma et al., 2005)	number of domains	hybrid	matchmaking
(Sapkota et al., 2005)	number of nodes	hybrid	matchmaking
(Paolucci et al., 2003)	number of nodes	P2P	matchmaking

3.5.5 Recoverable Information

This functional criterion identifies which types of information a discovery system supports for describing services. The values that this criterion accepts are: *business*, *functional*, *non-functional* and *meta-model*. The *business* value indicates that a system supports

Table 3.4: Discovery approaches with respect to standard conformance.

Evaluated approach	UDDI	WSDL	WSDL-S	WSMO	OWL-S
(Ran, 2003)	X				
(Makris et al., 2006)	X				
(Zhou et al., 2004)	X				
UDDI	X				
(Wang and Stroulia, 2003)		X			
(Schmidt and Parashar, 2004)		X			
(Dong et al., 2004)		X			
(Sivashanmugam et al., 2003)		X	X		
(Verma et al., 2005)		X	X		
(Cardoso and Sheth, 2003)		X	X	X	X
(Jian and Zhaohui, 2005)	X	X			
(Kokash et al., 2006)	X	X			
(Lee et al., 2007)	X	X			
(Zhuge and Liu, 2004)	X	X			
(Platzer and Dustdar, 2005)	X	X			
(Sapkota et al., 2005)				X	
(Paolucci et al., 2003)					X
(Kawamura et al., 2005)					X

provider's descriptions, such as location, contact details, etc. The *functional* value is intended to evaluate if a system supplies consumers with a description of *what* a service does. Instead, the *non-functional* value concentrates upon *how*, i.e., this value indicates that a discovery system supports non-functional descriptions of available services. It is worth noting that the non-functional value is not associated with the quality attributes of the discovery system under characterization. Finally, the *meta-model* value describes whether an approach supplies the definition of the semantics of service descriptions. These values are not exclusive, e.g. the recoverable information of UDDI is "business" and "functional".

Functional descriptions are necessary for communicating what a service does. Non-functional characteristics become important when there are, at least, two services designed for performing the same task. Indeed, having a description of how this task is carried out allows discoverers to select the service that best fits their QoS requirements. On the other hand, meta-model descriptions are essential in automatic environments.

However, these require to place extra effort on building domain ontologies and annotations.

3.5.6 Query language

This functional criterion deals with how easy is to formulate a query and how expressive a query language is. The values that this criterion accepts are: *keyword*, *natural language*, *constraint*, *template* and *meta-model*. The *keyword* and *natural language* values are intended to describe syntactical keyword-based and textual querying approaches, respectively. The *constraint* value is used for referring to languages that accept assertions, like "*availability greater than 0.9*". The *template* value is intended to describe languages that accept service descriptions as queries. For example, Woogle (Dong et al., 2004) expects a WSDL document as a query, so that its query language has been evaluated as being "template". Finally, the *meta-model* value focuses on languages that allow discoverers to query based on the definition of service descriptions. Also, these values are not exclusive, for example the query language of Matchmaker (Kawamura et al., 2005) has been evaluated as being "keyword", "natural language", "constraint", "template" and "meta-model".

The implication of the type of query language supported is twofold. In the first place, different query languages enable achieving different degrees of accuracy in search results. Qualitatively, the search results achieved by using annotations are better than the results achieved by keyword-based approaches, because semantic-based query languages allow discoverers to precisely state their needs. In the second place, in the case of constraint-based and semantics-based queries, modeling a need may require developers to learn a new query language.

3.5.7 Mediation support

This functional criterion evaluates if a discoverer system supports mediation. Conceptually, mediation attempts to bridge different representations of the same concept. In the context of Web Service discovery, mediation occurs between the representations of service descriptions and queries. The values that this criterion accepts are: *none, syntactic mediation*, *concept alignment* and *ontology alignment*. The second value is intended to describe when syntactic exact matching is complemented with mediation, like bridging synonyms –words that represent the same concept but are syntactically different from each other–, bridging different WSDL styles for specifying data-types, clustering parameter names that usually co-occur, and partially matching a bag of keywords.

The semantic matching value refers to mediation between two concepts from an ontology. This value is defined for identifying those approaches exploiting semantic relations (e.g.,

"same as", "sub-class of", "inverse of") among concepts, such as the semantic matching algorithm presented in (Paolucci et al., 2002). The ontology alignment value represents those approaches that mediate between two or more ontologies.

The mediation support characteristic carries several implications. If a discovery system does not supply discoverers and publishers with mediation (the *none* value), both must strive to accurately describe their needs and advertisements respectively, to produce unambiguous representations. Consequently, connecting services to consumers becomes harder. Although the *concept alignment* and *ontology alignment* values seem similar, the difference is that the former value requires providers and consumers to agree on a shared ontology. Instead, the second value allows users to bridge different conceptualizations of the same domain.

3.5.8 Results granularity

The results granularity criterion differentiates between approaches returning a service description and those returning an operation description. Here, a service description consists of a set of operation descriptions. Accepted values for this criterion are: *service* and *operation*. It is worth noting that the *service* value does not subsume the *operation* value, thus these values are not mutually exclusive.

The granularity of the results is important for invoking discovered services because an operation is is the smallest piece of software that a discoverer may invoke. Suppose a Web Service that provides classic e-mail operations, such as validate addresses, filter spam and send anonymous messages. If this service is retrieved when looking for a spam filter and the granularity of the results is *service*, a discoverer must crumble the description of the retrieved service to obtain the proper operation. Instead, *operation* results granularity allows consumers to directly invoke the service operation. However, some services present order dependencies among their operations. For example, non-free of charge Web Services often require to invoke an operation for checking account permissions, before invoking any other offered operation. Then, for some services it is essential to obtain the whole service description.

Table 3.5 shows the functional analysis results of the 21 described approaches.

3.6 Discussion

The presented survey of ongoing approaches to service discovery allows identifying:

1. some characteristics that are more popular than others within discovery approaches,

2. four dependencies among the evaluated aspects, and

3. major research possibilities in the field.

The characterization results show that some characteristics are more popular than others. For instance, most of the discovery systems follow the UDDI and WSDL specifications (see Table 3.4), support queries based on keywords and natural language descriptions (see Table 3.5), manage functional and business data, and follow a centralized matchmaking approach (see Table 3.3).

The popularity of these characteristics may be probably related to three facts. First, the design of many discovery systems might be influenced by the conception of a registry that was proposed with UDDI. Second, probably because that Web Service standards are text-oriented, many discovery systems adapt IR techniques, which are based on a centralized VSM and enable natural language queries. Third, IR-based approaches can be transparently adopted. The implication of the popularity of these characteristics is that the designers of service-oriented systems find more alternatives having these popular characteristics, when they look for a discovery system.

Returning to the evaluation results, they allow the identification of four dependencies among the proposed criteria by overlapping the functional and non-functional characterizations. First, the scalability and availability of many approaches are limited to the number of nodes and federated values, respectively. Commonly, these approaches depend on the UDDI specification. For example, Zhuge and Liu (2004) employ a three layers architecture, wherein the lowest layer is an UDDI registry. Likewise, (Ran, 2003) and (Al-Masri and Mahmoud, 2007) extend the UDDI specification, whereas the former employs decentralized "certifiers" and the latter a brokering system.

The second identified dependency among the proposed criteria, is that there is a correspondence between the foundations of IR-based approaches and the non-functional characterization results, which is shown in the column Scalability of Table 3.3. The reason behind this fact is that IR-inspired approaches, such as (Wang and Stroulia, 2003; Platzer and Dustdar, 2005; Birukou et al., 2007), employ a weighting technique –named TF-IDF– that has to be recomputed when new services are published in these systems. Intuitively, this characteristic may harm the performance of these approaches on dynamic and massively distributed environments, in which updates occur frequently and the VSM is shared among several distributed peers. Moreover, most of the analyzed IR-based approaches operate by matching a query against the vector representations of all available services, which requires more matching operations as the number of published services grows.

Third, it would be rational to expect that approaches supporting non-functional descriptions of services rely on brokers, because they are essential to collect reliable QoS informa-

tion about providers. In spite of this, most of the alternatives that support non-functional information adopt a matchmaker approach. Instead, (Al-Masri and Mahmoud, 2007) employs a broking system.

The fourth found dependency appears by analyzing the results shown in Table 3.5 and Table 3.4 together. There is a clear correspondence between UDDI and WSDL standards and IR-based approaches. IR-based approaches have shown that they can be perfectly adapted to deal with Web Services, mainly because both UDDI and WSDL are text-oriented. Migrating from an UDDI-based platform to an IR-based counterpart, which should grant more query language facilities to discoverers, may not demand severe modifications. A fact that supports this claim is the large number of publicly available WSDL documents that have been used for evaluating IR-based approaches. On the other hand, if an approach follows WSDL then it should be easy to adopt the WSDL-S and, in turn, to incorporate annotations.

Additionally, this survey of ongoing approaches to service discovery points out major opportunities for research. There are two research challenges related to UDDI and WSDL complaint discovery systems. Being UDDI and WSDL complaint means that a Web Service description consists of an entry in a UDDI registry, and a WSDL document. The entry in a UDDI registry contains meta-data to support discovery, in which an important part of such an entry is a category associated with the Web Service being published. Moreover, usually keywords present at UDDI entries and WSDL documents are used to support discovery. Then, the descriptiveness of these keywords is very important as well. Despite the importance of classifying services into categories and conveying explanatory keywords within WSDL documents, in practise UDDI Yellow Pages are scarce, and WSDL document keywords are not as explanatory as one might expect (Fan and Kambhampati, 2005).

Assigning a proper category to a service can be a tedious and error prone task due to the large number of categories usually present in Web Service registries. Commonly used taxonomies such as the UNSPSC or the SIC contain hundreds of categories. Thus, some researchers have strove to ease Web Service classification. However, previous efforts for assisting publishers to classify Web services have several shortcomings.

MWSAF (Patil et al., 2004) is an approach for classifying Web Services, which translates formal descriptions of real world categories and argument definitions into a graph. Then, graph similarity techniques are used for comparing both. Likewise, Duo et al. (2005) translate a definition into an ontology, instead of into a graph. Then, an ontology alignment technique attempts to map one ontology onto another. The main limitation of these matching approaches is that they do not attempt to reduce the distance among different coding conventions or encapsulation styles that are usually present in Web Service argument definitions. In fact, they achieve low accuracy, which is shown in their

experimental evaluations (Patil et al., 2004; Duo et al., 2005).

METEOR-S (Oldham et al., 2004) describes a further improved version of MWSAF, in which the graph matching technique is replaced with a Naïve Bayes classifier. To do this, METEOR-S extracts the names of all operations and arguments declared in WSDL documents of pre-categorized Web Services. The main limitation of this approach is that it assumes independence between the name of an operation and its arguments. Clearly, the name, or header, of an operation and the name of its arguments might be related, e.g., print(Style, Document) method signature. Therefore, this classifier seems to be based on a false premise. Moreover, though METEOR-S proposes a document classification approach, natural language documentation, usually present in WSDL files and service registries, is not considered. However, METEOR-S experimental results shows an accuracy improvement with respect to MWSAF for the same data set.

Assam (Heß et al., 2004) is an ensemble machine learning approach that combines the Naïve Bayes and Support Vector Machine algorithms to classify WSDL files in manually defined hierarchies. Assam takes into account comments present at WSDL documents and UDDI entries. As reported, this approach is more accurate than similar approaches, even though its authors used a repository of 391 Web Services (their WSDL documents and textual comments published in the associated registry) divided into 11 categories for experimenting. This repository[4] has been made publicly available, which by itself has been an important contribution to the field. The accuracy of the approach was evaluated with a tolerance value. A tolerance value of t represents that the correct classification is included in a sequence of $t + 1$ suggested categories. The evaluation shows that when suggesting a domain for annotating a service the accuracy was 60% for a tolerance value of $t = 0$, and it trended towards 90% for $t = 10$.

The problem associated with analyzing the descriptiveness of WSDL document keywords has been partially addressed in current literature. Regarding documentation present in such documents, Fan and Kambhampati (2005) show that in the 80% of 640 real-life WSDL documents the average documentation length for operations is 10 words or less. Furthermore, from this 80%, half of them have no documentation at all.

Pasley (2006) explains the impact of using general purpose data-types on the maintainability and discoverability of Web Services. The author refers to XSD special constructors that allow defining data-types capable of exchanging any XML content as "wild-cards". Although using wild-cards creates vague interface contracts without explanatory keywords, Pasley detects that one possible reason for using them is to minimize the effort involved in modifying a service when it evolves, while assuring that consumers bound to old versions of the service will be able to correctly invoke and process the operations defined in its new version.

[4]Categorized Web Services Repository, `http://msi.ucd.ie/RSWS`

Likewise, Blake and Nowlan (2008) detect poor naming tendencies in WSDL documents and empirically show that the performance of a discovery system was improved after dealing with the observed tendencies. Broadly, the authors show that developers use common phrases within parameter part names, abbreviations and names shorter than three characters, which are ineffective for matching part names and queries.

Alternatively, as the keywords conveyed in queries are also an essential part for textual-oriented discovery systems in general, and IR-based in particular, query generation presents research opportunities as well. Thus, in 2006 researchers started to evaluate whether extracting tacit knowledge from clients' service-oriented applications may contribute to build accurate queries, which in turn may help to increase the precision of the underlying service discovery mechanisms.

Following this line, in (Dourdas et al., 2006) the authors preprocess natural language descriptions of functional requirement specifications, in order to gather explanatory keywords. The authors' thesis is that many software documentation elements may represent additional sources of relevant keywords for enhancing the description of users' requests. Besides employing stemming and word sense disambiguation, the authors show a technique for matching built queries to pre-defined patterns of queries. The authors propose to maintain associations between queries and the candidates selected by the discoverer who used such queries, to take advantage of past users' experiences.

Blake et al. (2007) also promote the idea of extracting information from client applications and using it for creating queries. The authors employ a software agent (Wooldridge and Jennings, 1996) for assisting a developer in finding Web Services based on knowledge of the development environment. Basically, this agent periodically monitors the developer until it detects an action that may be associated with requesting a service. The agent then uses any captured textual input and contextual information (e.g., the name of the project the user is working on, the developer's role, etc.) to search service repositories. When a relevant service is discovered, the agent presents the results to the user, who must decide what to do with the service (options are to execute it, not to execute it, or defer the decision). Then, the agent gradually infers the user's preferences with regard to whether a retrieved Web Service should be used or not. The uttermost goal of the approach is to automatically execute or discard services in new and similar situations.

Discovery systems based on semantics present two research challenges, at least. These approaches lack their vital inputs, namely ontologies and semantically annotated Web Services (McCool, 2006). To make Semantic Web Services a reality, it is necessary to define standard domain ontologies for describing services. In parallel, more effort should be placed on developing semi-automatic tools for annotating Web Services, to make publishers' tasks easier. Alternatively, annotated Web Service descriptions may be influenced

by the recent Web 2.0 and Social Web successes (McCool, 2006). For instance, Seedka![5] is a repository of "tagged" services that announces 27.388 annotated services at the moment of writing this thesis. Tags are keyword-based annotations without the notion of synonyms or disambiguation, but these are easy to create, share and use. Tag lightweightness with respect to ontology definition languages, encourages developers to annotate their services. The idea of annotating service descriptions with lightweight semantics has been also explored in (Kiyavitskaya et al., 2009).

Another identified challenge relates to the importance of trustworthy Web Service descriptions in the context of automatic discovery. Automatic discovery of services requires that discoverers' can trust on providers' descriptions. Wang et al. (2008) state that just because the Internet is a very open heterogeneous environment, the SOC paradigm must face a major challenge, namely, the trust between service consumers and providers. This is an interesting problem that presents many opportunities for research, which not only comprise how to ensure classic non-functional QoS requirements such as response time or availability, but also functional aspects –determine services functionality– and legal ones. Besides ensuring classic non-functional QoS requirements, this problem comprises vetting the services to determine their functionality, safety –lack of malware– and legal aspects, such as licenses and copyright violations. (Wang and Vassileva, 2007) discusses current trust and reputation mechanisms that have been applied to Web Services.

3.7 Conclusions

Discovering Web Services in the Internet has been widely recognized as a very difficult task. Therefore, different approaches to alleviate service discovery have been proposed in the literature. Previous sections have analyzed some of the most relevant approaches to the purpose of this thesis. Unfortunately, the approaches discussed do not cope with a subset of the problems that are essential to truly enable the discovery of services on environments that manifest an incessantly increasing number of services. Namely, these problems are:

- *Retrieval effectiveness versus cost of adoption trade-off*: there is a lack of effective techniques for discovering services that, at the same time, do not impose heavy costs on publishers. Semantics-based approaches, for example, achieve high levels of retrieval effectiveness, which surpass the effectiveness achieved by syntactic-based ones. However, as the backbone of semantic approaches is describing each detail of publicly available services using machine interpretable languages, publishers must put extra effort into describing services by means of semantic meta-data.

[5] Seedka!, `http://seekda.com/`

With search systems in general, the more effort we put into accurately describing our goods, the better retrieval effectiveness we get. This situation involves gaining retrieval effectiveness, at the expense of making service publication rather hard. However, in order to alleviate service publication and, at the same time, achieve better retrieval effectiveness, lightweight and effective approaches to service discovery should spring.

- *Incorrect usage of the meta-data that support Web Services discovery*: many problems related to the efficiency of standard-compliant approaches to service discovery, may stem from the fact that current Web Service standards to describe services are incorrectly or not fully employed by publishers. These standards provide means for describing offered services from a functional perspective, and allow publishers to advertise their services using catalogs, or taxonomies. Although the intuitive importance of properly advertising services, the usage of service taxonomies is scarce in practice. In addition, some practices that attempt against the discoverability of services, such as poorly documenting WSDL documents or using unintelligible naming conventions, are frequently found in publicly available Web Service descriptions. Therefore, to overcome some of the problems related to the efficiency of approaches based on Web Service standards, guidelines to make better advertised and more discoverable Web Services should arise.

- *Limited support to describe what a discoverer is looking for*: similarly to the aforementioned trade-off, there is a lack of powerful means to describe information needs without requiring heavy query descriptions or all the specifications of full semantic techniques. Web Service standards and IR-based approaches to service discovery provide merely keyword-based query support. Due to the heterogeneity of the Internet, and massively distributed environments in general, different keywords are commonly used for advertising services that offer the same functionality. In practice, this occurs because different services are built by different development teams, who might share neither the same programming conventions nor the same first language. The inconsistency between keywords in interfaces of publicly available services and queries, along with the intuitive limitations of this kind of queries to describe users' intentions, may be the cause of many problems related to the retrieval effectiveness of lightweight approaches to service discovery. To overcome these problems, there is a need to explore novel methods to describe users' requests and match them to publicly available service descriptions.

- *Missing plans to face the "curse of success"*: as Web Service technologies become massively adopted, the number of published services is expected to grow. The success of Web Services and SOC in becoming the next big thing in the software industry, lies on the ability of service discovery systems to manage an ever increasing

number of publicly available services. Recently, some researchers have striven to explore the ability of different service registries to deal with a growth of the number of advertised services. Obviously, studying the scalability of the infrastructure that supports the discovery process is a very important concern in this direction, however it is not the only direction worth exploring. Discoverers' frustration with search results, often increases with the space of published services. There is a lack of search space reduction techniques, which allow discoverers to promptly obtain proper Web Services, in spite of the huge number of alternatives. Therefore, more effort should be put into improving the scalability of discovery systems, without neglecting their retrieval effectiveness.

Tacking into account the problems listed above, it is clear that the existing standards-based service discovery technologies are insufficient for building a global service infrastructure. The next chapter describes an approach to solve these problems.

Table 3.5: Functional characterization of Web Service discovery systems.

Evaluated approach	Recoverable Information	Query language	Mediation support	Results granularity
UDDI	business, functional	keyword	none	service
(Zhuge and Liu, 2004; Schmidt and Parashar, 2004)	business, functional	keyword	syntactic mediation	service
(Wang and Stroulia, 2003; Jian and Zhaohui, 2005; Platzer and Dustdar, 2005; Kokash et al., 2006; Lee et al., 2007)	functional	keyword, natural language	none	service
(Dong et al., 2004)	functional	template	syntactic mediation	service, operation
(Birukou et al., 2007)	functional	keyword, natural language	syntactic mediation	operation
(Ran, 2003; Al-Masri and Mahmoud, 2007)	business, functional, non-functional	keyword, constraint	none	service
(Zhou et al., 2004)	business, functional, non-functional	keyword	none	service
(Makris et al., 2006)	business, functional, non-functional	template	none	operation
(Kozlenkov et al., 2007)	functional, non-functional	template, constraint	none	operation
(Kawamura et al., 2005; Paolucci et al., 2003)	business, functional, non-functional, meta-model	keyword, natural language, constraint, template, meta-model	semantic matching	operation
(Cardoso and Sheth, 2003)	functional, meta-model	natural language, template, meta-model	ontology alignment	operation
(Sivashanmugam et al., 2003; Verma et al., 2005)	functional, meta-model	template, meta-model	semantic matching	operation
(Sapkota et al., 2005)	meta-model	meta-model	ontology alignment	service

Chapter 4

EasySOC

Service discovery is an essential activity for the development of service-oriented applications. However, service discovery presents many challenges that make the SOC paradigm hard to implement. Having revised current state of the art, there is a clear need of novel approaches to efficiently and accurately discover services in the Internet, which at the same time do not increase the effort destined to service publication and discovery. In current approaches to service discovery there is a correlation between developers' effort and the benefit they may obtain from a discovery system. In other words, the more effort developers put on describing their services and service needs, the more accurate discovery results they would obtain. Due to the complexity related to publication/discovery, such approaches are rarely employed in practice, which is one barrier to the emergence of worldwide software component marketplaces (Hummel et al., 2008).

This research strives to overcome the problems associated with service discovery hindering the development of service-oriented applications. The cornerstone of this research is the following thesis:

Thesis It is feasible to allow discoverers to efficiently outsource third-party services, without neither requiring all the specifications of full semantics-based techniques nor charging them with the task of defining highly descriptive and precise queries.

The idea conveyed within the thesis is that methods to service publication and discovery lighter than those based on semantics, can be a feasible way towards the materialization of service-oriented applications. To do this, this research seeks three main goals, one regarding publication and the others regarding discovery.

Goal 1 With respect to publication, the point of this research is to assist publishers to make more discoverable services without requiring to make exhaustive annotations.

Goal 2 With respect to discovery, the main goal of this research is to allow discoverers to retrieve proper services without requiring heavy query descriptions.

Goal 3 This research also aims to allow discoverers to retrieve proper services without imposing execution time overheads to the discovery process.

In order to limit the scope of this work, it is focused on techniques for facilitating the publication and discovery of Web Services, because these have been mostly employed to implement the SOC paradigm (Erickson and Siau, 2008). Therefore, this document presents an approach that can be used with current technologies and standards. The rest of this chapter provides more details about the approach.

4.1 Overview of the proposed approach

Central to the proposed approach is the fact that properly described Web Services and consumers' applications, i.e., client-side applications designed to consume third-party services, may convey information for bridging the gap between services and discoverers. Specifically, the WSDL document and UDDI entry of a service may convey relevant information about its offered functionality, like the name and comment of each offered operation, or the category of the service, which may help to enhance its discoverability. At the same time, the source code associated with client-side applications may carry relevant information about the functional descriptions of the potential services that can be discovered and, in turn, consumed from within those applications. This information may help to improve the descriptiveness of the consumers' outsourcing intentions.

Therefore, the outline of the approach proposed in this work is to gather relevant information from the descriptions of services and consumers' applications, then the gathered information is represented in such a way that it enables precise and prompt service discovery.

Relevant information Any word that may help to infer service functionality or consumers' outsourcing intentions.

The cornerstone of the proposed approach is to exploit current standards and existing resources, instead of requiring new service and query descriptions or putting effort into adding semantics. As publication and discovery are essential activities of the SOC paradigm, and this approach aims to ease these activities, it is called "EasySOC".

During the development of the EasySOC approach, some challenges have been undertaken. For the sake of readability these challenges can be listed as follows:

1. Gathering relevant information from WSDL documents and UDDI entries.
2. Gathering relevant information from consumers' applications source code.
3. Representing gathered information while enabling efficient service discovery.

Regarding challenges 1 and 2, in practice the aforementioned relevant information is neither well structured nor clearly stated nor explanatory. With regard to challenge 3, though relevant information were gathered, representing it in order to enable precise and prompt service discovery is also another big challenge.

The WSDL and UDDI specifications are well structured, which allows gathering relevant information from them effortlessly. However, relevant information carried within WSDL documents, is commonly concealed behind technological aspects, such as transport protocols, bindings and data encoding details. Even worse, publishers often build unintelligible service descriptions that do not convey names and comments relevant to the discovery process. Therefore, unveiling information about a service functionality from its WSDL document can be a hard task, if not impossible, for instance when publishers produce "cryptic" service descriptions containing irrelevant data. Likewise, UDDI Yellow pages are scarcely populated in practice because assigning a proper category to a service can be a tedious and error prone task due to the large number of categories usually present.

On the other hand, the information related to consumers' outsourcing intentions can be found within source code files, wherein it is covered by portions of code that are irrelevant to the discovery process of required services.

To overcome the challenges associated with gathering relevant information from WSDL documents and UDDI entries, this work presents novel guidelines to help publishers on making services that convey relevant information and a novel way for classifying them. To assist publishers in the creation of more discoverable services, common practices that produce non-descriptive Web Service descriptions have been identified, and novel solutions to them have been built as part of this work. Accordingly, this thesis presents a novel catalog of Web Service discoverability anti-patterns as a main part of the EasySOC approach.

Discoverability anti-pattern Is a general reusable solution to a commonly occurring problem that attempts to prevent the discovery and understanding of any Web Service description.

In order to ease the classification of Web Services during publication, the EasySOC approach aims at automatically deducing proper categories for WSDL documents, according to their offered functionalities. The idea is to assist publishers by suggesting several candidate categories for the services they aim to publish.

Once properly described and classified services are available, so that the Goal 1 is accomplished, novel heuristics for automatically gathering relevant information from such descriptions are presented as another main part of the EasySOC approach. Broadly, these heuristics aim to exploit the structure of the WSDL language and typical programming conventions, for gathering names and comments of service port-types, operations, messages and data-types.

To overcome the challenge associated with gathering consumers' outsourcing intentions, the idea followed by this work is that developers should be focused on building their applications, while automatic heuristics pull out information standing for consumers' queries from such applications. Then, as another main part of the EasySOC approach, these heuristics combine common practices for developing service-oriented applications with the ideas of *Query-by-Example* and *Query Expansion*. To understand the sound reasoning that supports this combination, the definition of Query-by-Example, Query Expansion, and an important characteristic found in most applications designed to consume Web Services, should be reviewed.

Query-by-Example Is an approach for creating queries that allows a discoverer to search for an entire piece of information based on an example in the form of a selected part of that information.

Query Expansion Is an approach for augmenting the quantity of relevant information within queries that allows discoverers to express their needs using only an specific part of relevant information.

An important characteristic of most service-oriented applications is that developers frequently include into their source code "black boxes" that internally stand for services that are going to be outsourced (Shapiro, 1986; Eugster, 2006). These "black boxes" provide programmatic descriptions of the functionalities of the needed services, i.e., they can be seen as examples of what discoverers are looking for. Then, the EasySOC approach exempts developers for making an additional effort for defining queries, by transparently pulling out relevant information from these examples. Moreover, extracted queries can be expanded, by gathering information from the source code representing internal components that directly interact with external services, i.e., the "black boxes", because expanding queries based upon components with strongly-related and highly-cohesive operations should not only preserve, but also improve, the meaning of the original request. To sum up, the EasySOC approach aims at allowing discoverers to describe their needs precisely, but also in an easy way, for achieving Goal 2 of this research.

Until now this section described how EasySOC deals with challenges 1 and 2, i.e. gathering information relevant for the discovery process from UDDI, WSDL documents, and

client-side source code. In order to face the challenge related to represent gathered information while enabling efficiently service discovery, another main part of the approach consists of an adaptation of the well-know Vector Space Model (VSM).

One of the premise of the EasySOC approach is to exploit all the gathered information about a published service, including its category. In fact, the category of available services is one of the most important elements of the VSM adaptation proposed here. Each available category is used for dividing or partitioning the vector space in sub-spaces. Vector representations of services belonging to a particular category are associated with the corresponding sub-space. Then, instead of comparing a query against a large number of service representations, the idea is to make a first reduction of the search space by deducing the category, or sub-space, of potential services.

The reduction of the search space can be interpreted as an automatic version of a typical manual discovery process, which usually comprises two steps: browsing available categories and selecting service from a selected category. As will be formally explained below, Goal 3 of this research can be accomplished since automatically reducing the search space makes Web Service retrieval computationally efficient even with many available services.

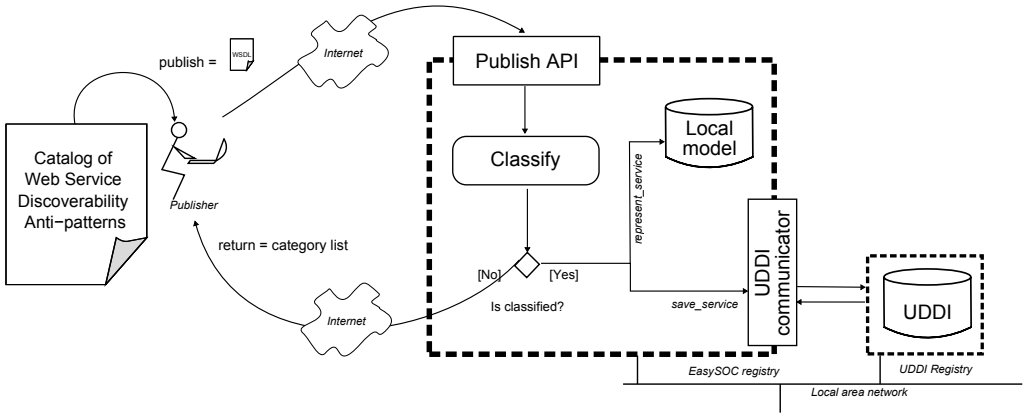

Figure 4.1: Publication assistance.

The aforementioned support have been integrated into a registry for Web Service descriptions that extends the UDDI specification, to prove that the core ideas of the EasySOC approach are workable and feasible. The registry is called EasySOC registry. Graphically, Figure 4.1 depicts the main activities and actors of the process that takes place when publishing Web Services through the EasySOC registry. The process starts when a publisher, on the left side of the figure, publishes a Web Service description in the registry. The published description consists of a WSDL document and UDDI data. As the publisher has been previously supplied with the catalog of anti-patterns, the bad practices should not be present in the WSDL. In the future, WSDL documents will be analyzed to find bad practices automatically. Then, the publisher receives a list proper categories, that

allows him/her to classify it, without having to browse the entire taxonomy of available domains. Instead, if a service is categorized relevant information is gathered from its description. The gathered information is represented and stored locally in the partitioned VSM, whereas the rest of the publication request is forwarded to an UDDI registry.

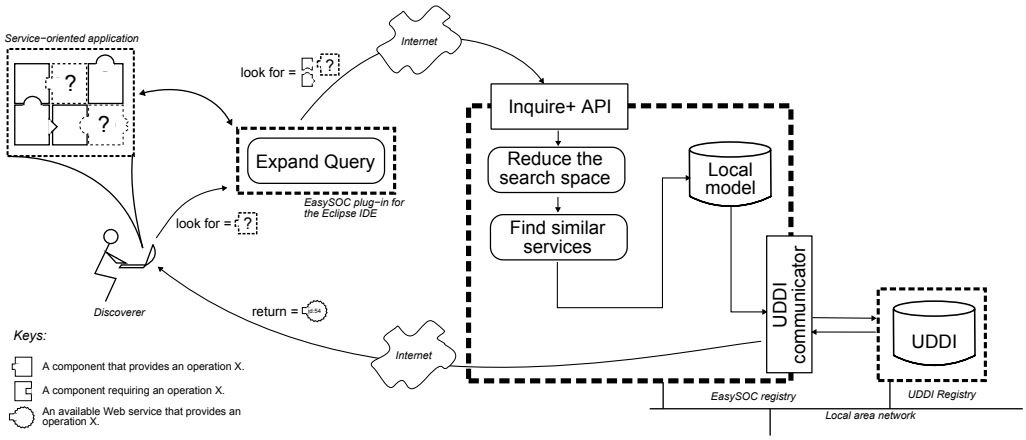

Figure 4.2: Discovery assistance.

Graphically, Figure 4.2 shows a discoverer querying the registry for services similar to a given example. In Figure 4.2 an example is sketched as a component of the target application using dashed lines, and represents the expected interface of a potential third-party service. Then, the example, or query, is expanded based on local components that directly interact with it. Local components are sketched using continuous lines. The query generation process has been implemented as a plug-in for the Eclipse IDE, and it is called EasySOC plug-in. Once the query has been built, it is matched onto the representations of information gathered from WSDL documents previously published. Occasionally, the example can be used to reduce the space of available services, by automatically filtering those services that do not belong to the category deduced for the example. Finally, WSDL descriptions of the potential services are retrieved from an UDDI registry and returned to the discoverer.

Next sections explain how to gather relevant information from UDDI and WSDL documents (section 4.2), and client-side source code (section 4.3). Afterward, section 4.4 describes how to represent such information while enabling efficiently service discovery. Then, in section 4.5 the catalog of Web Service discoverability anti-patterns is presented.

4.2 Gathering relevant information from Web Service descriptions

Text mining, also known as intelligent text analysis, refers to the process of extracting interesting and non-trivial information and knowledge from unstructured text (Hearst, 1999). EasySOC uses text mining for gathering relevant information from Web Service descriptions, because the information present in WSDL documents are mostly comments written by developers, as (Sabou et al., 2005) assert. In general, these comments are written in English, have a low grammatical quality, punctuation is often ignored and several spelling mistakes and snippets of abbreviated text are present. Moreover, in an open world setting, where services built by different organization are made available, the same abstract service can be described using different WSDL documents (Cavallaro and Di Nitto, 2008). As a result, two services conceived for a particular task may have a *syntactically* different interface, such as sendMail(ns:email e) and sendMail(xs:string sFrom, sTo, sSubject, sBody).

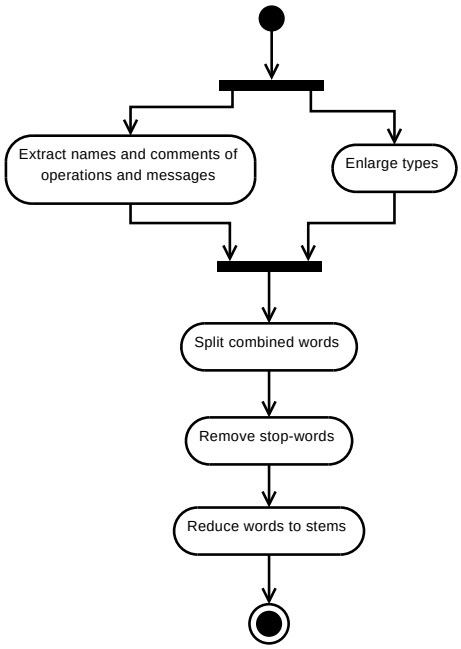

Figure 4.3: Text mining process for WSDL documents.

EasySOC proposes a novel text mining process that takes into account the aforementioned particularities of service descriptions. Figure 4.3 shows the text mining process, which receives a WSDL document as input an returns a collection of stems. The first two activities of the text mining process can be done simultaneously. The former activity involves extracting the name and textual documentation that are associated with each service operation and message. The latter activity is intended to gather relevant information

from defined data-types. In order to deal with the different encoding styles mentioned, if an argument data-type is non-primitive (i.e., is not an integer, string, boolean or float) the idea is to look for the corresponding definition and extract the names of the elements that are encapsulated in the composed type. A novel type enlargement algorithm is used to do this.

Formally, the type enlargement algorithm is described Algorithm 4.1. The algorithm receives an XSD type definition that combines some sub-types by using XSD encapsulation constructors like xsd:element, xsd:sequence, xsd:complex. By using these constructors a sub-type is associated with a name, then the algorithm returns these names. The type enlargement algorithm defines how to expand xsd:complex and xsd:element types, and then defines rules to break down other cases into these base cases. Type enlargement occurs at most at one level of depth, instead of fully recursively, limiting the overall complexity of the algorithm. Arrays of "something" are handled in the same way.

```
1:  procedure ENLARGE(type, follow)                    ▷ Returns a String[]
2:      if !follow then
3:          r ← type.name
4:      else if type is complex then
5:          for all s ∈ type.secuenceElements do
6:              r ← r + s.name
7:          end for
8:      else if type is element then
9:          for all c ∈ type.children do
10:             r ← r + ENLARGE(c, false)
11:         end for
12:     end if
13:     return r
14: end procedure
```

Algorithm 4.1: Type enlargement.

In a third activity, by splitting combined words the text mining process attempts to bridge different naming conventions. In general, developers combine a verb and a noun for denoting the name of an operation, such as getQuote or get_quote. Then, every distinct operation and message that follows each naming denomination would be treated as a different word. In order to bridge different naming conventions, the process looks for combinations of words and splits them into verbs and nouns. Combinations of more than two words are taken under account, e.g., for the combined word "getQuoteFor" the text mining process separately dumps the words "get", "quote" and "for". Table 4.1 summarizes the rules for splitting combined words.

In a fourth activity, by removing symbols and stop words the text mining process attempts to "clean" service descriptions. A stop word is a word with a low level of "use-

Table 4.1: Rules for splitting combined words.

Notation	Rule	Source	Result
Java Beans	Splits when changing text case.	getZipCode	get Zip Code
Hungarian	Splits when changing text case.	ulAccountNum	ul Account Num
Special symbols	Splits when either "_" or "-" occurs.	get_Quote	get Quote

fulness" within a given context or usage. A list of about 600 English stop words and a small list of stop words related to the Web Services domain, such as request, response, soap and post, are employed by the text mining process.

Finally, the text mining process ends by executing the Porter Stemming algorithm (Porter, 1997) for removing the commoner morphological and inflectional endings from words. Thus, the output of the text mining process are stems of English words, which EasySOC uses for building a vector $\vec{v} = (e_0, ..., e_n)$, where each element e_i represents the weight of a distinct stem for the WSDL document as is explained in section 4.4.

For the sake of exemplification, next is shown the result of employing the text mining process on the WSDL document of Figure 2.5. Table 4.2 shows two collections of word occurrences. The collection on the left, contains the extracted words along with their occurrences (only applying the first step of the text mining process). Conversely, the collection on the right contains the stems, and their occurrences, generated by applying all the steps of the text mining process. By utilizing the text mining process relevant information is gathered. For example, by preprocessing the message "GetRateSoapOut" the stop words "get", "soap" and "out" are removed, while the relevant word "rate" arises. On the other hand, by enlarging the data-type associated with the message "GetRateSoapIn" the stems "src", "currenc", "dest" and "currenc" are derived from the combined words "srcCurrency" and "destCurrency".

Table 4.2: Text mining process example.

Collection of extracted words (Step 1 of the text mining process)	Collection of stems (Steps 1 to 5 of the text mining process)
<GetRate;2>, <srcCurrency;1>, <destCurrency;1>, <GetRateSoapIn;1>, <CurrencywsSoap;1>, <This;1>, <method;1>, <returns;1>, <the;1>, <currency;1>, <conversion;1>, <ratio;1>, <between;1>, <two;1>, <countries;1>, <GetRateSoapIn;1>, <GetRateSoapOut;1>	<currenc;4> <rate;3> <method,1> <conver,1> <ratio;1> <countr;1> <src;1> <dest;1>

4.3 Gathering relevant information from source code

Anatomically, a service-oriented application can be viewed as a component-based application that is created by assembling two types of components: *internal*, which are those locally embedded into the application, and *external*, which are those statically or dynamically bound to a service. Central to this research is the idea of automatically gathering relevant information from the programmatic specifications of an interface describing the component being outsourced. This idea follows a metaphor that is known as Query-By-Example. This, besides avoiding the problem of knowing which low-level Web Service features are important for a given query, exploits human pattern matching capabilities.

Moreover, this work refines the Query-By-Example idea by gathering relevant information that can be found in the context in which the intended functionality is used, i.e. those internal components that directly interact with the external one. This refinement bases on Query Expansion and it is supported by the fact that following good practices when building component-based software results in components with strongly-related and highly-cohesive operations (Vitharana et al., 2004). Based on this fact and the definition of Function Cohesion provided by Papazoglou and Heuvel (2006), it is feasible to assume that the logic of a well-designed internal component commonly belongs to a unique domain which, in fact, is the same domain of its external service/s.

Functional Cohesion Cohesion is the degree of the strength of functional relatedness of operations within a service. The operations in the services must be highly related to one another, i.e. highly cohesive (Papazoglou and Heuvel, 2006).

For example, a Web Service for providing current foreign exchange rates may be useful for internal components belonging to the business domain, while it rarely might be useful for a component in the text processing domain.

In order to provide a down-to-earth explanation of Query-By-Example and Query Expansion for Web Service discovery, it is essential to understand the structure of mostly component-based service-oriented applications. The development of such kind of applications relies on libraries (e.g., CXF, JWSDP[1] and WSIF (Duftler et al., 2001)), which provide programming abstractions to keep the application code as clean as possible from Web Service communication details (Cibrán et al., 2007). Basically, the approach behind these libraries encourages developers to generate client-side code for representing the interface to an external service, i.e., its operations and argument data-types, and associate internal components with concrete proxies of this abstract service description. Similarly, the Spring framework (Johnson, 2005) provides a number of built-in proxies that can be injected into applications in order to easily represent services, provided they adhere

[1] Java Web Services Development Pack, `http://java.sun.com/webservices/jwsdp/index.jsp`

to a local interface. In both cases, proxies are in charge of shielding application logic from all non-functional activities, e.g., transmitting exchanged data over a network. In consequence, the anatomy of client-applications consists of internal components –a.k.a. dependants– invoking external services through classes standing for interfaces, arguments and proxies, as shown in Figure 4.4.

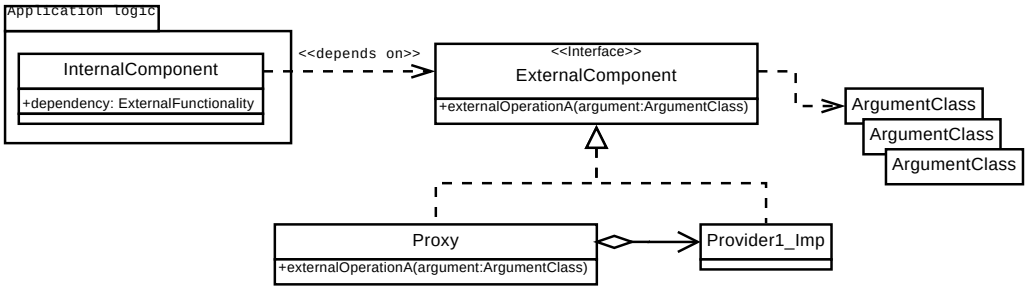

Figure 4.4: Anatomy of applications consuming Web Services.

The source code of the internal component depends on the interface, even though at run-time it will be bound to the proxy. Therefore, the interface named *ExternalComponent* stands for a programmatic description of a component being outsourced, while argument and dependant classes describe the context in which it is used.

The specification of an interface standing for a service consists of the signatures of its exposed operations (mandatory) and a textual description of its constituent parts (optional). In addition, the classes representing the arguments of these exposed operations and components that directly interact with them, may have proper names and documentation also. All in all, the information present in an interface describing a third-party service or in those components that invoke it, are mostly comments written by developers, and names of classes, methods, and arguments.

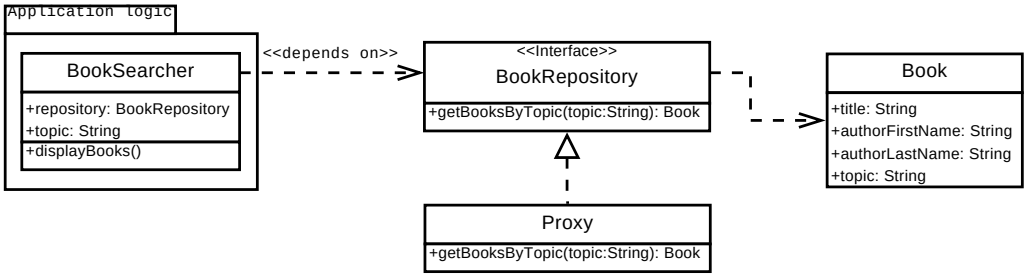

Figure 4.5: Class diagram of the book searcher example.

Having presented the common anatomy of mostly service-oriented applications, the following paragraphs will illustrate a novel text mining process to gather relevant information about consumers' outsourcing intentions. Let us suppose we want to incorporate a Web Service for obtaining information about books covering a topic into an applica-

tion written in Java. The class implementing the internal component asks the service for books of a given topic and then iterates the results to display the associated information. The detailed design of the implementation is shown in Figure 4.5.

The Java code implementing the book searcher is:

```
/** Looks for books of a particular topic and display them afterward */
public class BookSearcher {
        BookRepository repository;
        String bookTopic;
        public void displayBooks() {
                List<Book> results = repository.getBooksByTopic(bookTopic);
                // Display results
        }
}
```

The BookSearcher class uses an instance of BookRepository. As has been mentioned earlier in this section, commonly the service functionality is abstractly described by an interface, and the invocation is done through a concrete proxy. The implementation of the BookRepository interface is:

```
/** Retrieve books of a given topic */
public interface BookRepository {
        public List<Book> getBooksByTopic(String topic);
}
```

The BookRepository interface has one method that receives a topic and returns a list of instances of Book. The simplified class for representing book information comprises the following properties: title, authors, isbn and topic. Its implementation is:

```
/** A written work or composition that has been published */
public class Book {
        private String title;
        private String isbn;
        private String authors;
        private String topic;
}
```

For the sake of clarity, constructors and methods for accessing class properties have been omitted. As the reader can note, good naming and documentation practices were followed throughout the implementation of the example. In particular, there are comments at the header of the classes and explanatory names were employed.

As part of the EasySOC approach, this thesis presents a novel text mining process comprising five activities for gathering relevant information from the source code of service-oriented applications. Each cell of Table 4.3 shows the output generated by one individual activity of the text mining process when a particular source code was given. As shown in the columns of the table, the three initial activities enlarge the collection of extracted terms, while as long as the process goes on, the resulting collection of terms is refined. The rows of the table show that the query expansion approach quantitatively

Table 4.3: Output of each preprocessing activity.

Class/Activity	1^{st}	2^{nd}	3^{rd}	4^{th}	5^{th}
BookRepository *(interface)*	BookRepository getBooksByTopic	Retrieve books of a given topic	Book Repository get Books By Topic Retrieve books of a given topic	book repository books topic retrieve books topic	book repositori book topic retriev book topic
BookSearcher *(dependant)*	BookSearcher repository bookTopic displayBooks	Looks for books of a particular topic and display them afterward	Book Searcher repository book Topic display Books Looks for books of a particular topic and display them afterward	book searcher repository book topic display books looks books topic display	book searcher repositori book topic displai book look book topic displai
Book (*argument*)	Book title isbn authorLastName authorFirstName topic	A written work or that has been published	Book title isbn author Last Name author First Name topic A written work or that has been published	book title isbn author author topic written published	book titl isbn author author topic written publish

improves the resulting query (see row *interface* versus rows *dependant* and *argument*) by adding terms (stems to be precise) belonging to the "book" domain mostly.

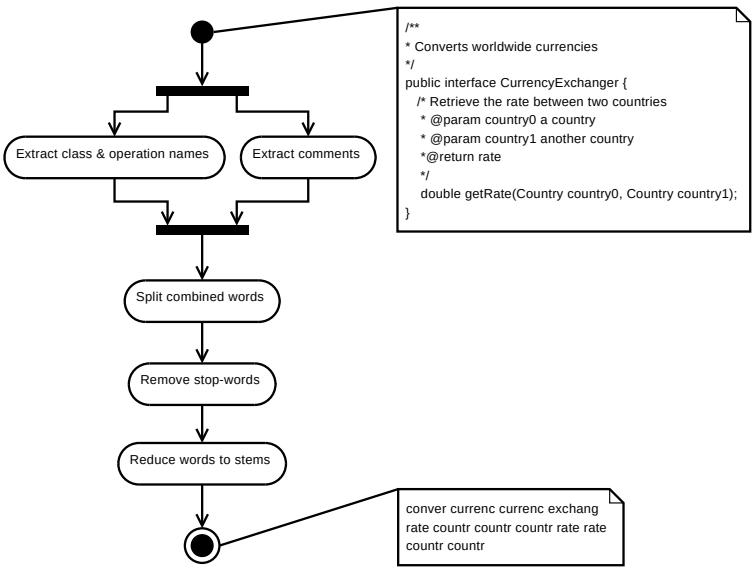

Figure 4.6: Text-mining process for service-oriented applications source code.

Figure 4.6 depicts the text mining process. In general, the process takes any component specification (a Java class in this case) as input and generates a collection of extracted stems. The first activity pulls out the name of a component and the name of its operations. Simultaneously, another activity gathers developers' comments. At this point the text mining process has a collection of words that may contain combined words, because commonly used notation conventions (e.g., JavaBean, Hungarian) suggest to combine words by means of either one or more uppercase characters, numbers or hyphens (e.g.,

getRate, *country0*, *get_rate*). Then, remainder activities are the same as the last three activities of the text mining process for WSDL documents, i.e. splitting combined words, removing symbols and stop words, and reducing English words to their stems. As a result, the output of the text mining process is a collection of stems extracted from a component source code.

In order to build queries, the text mining process of Figure 4.6 is employed by performing Algorithm 4.2. Steps 2 and 3 of the algorithm consist in identifying classes describing dependants and arguments (inputs and outputs indistinctly) corresponding to a given example. Here, the example given as input stands for an external service. Then, at step 7 the algorithm feeds the text mining process of Figure 4.6 with the classes identified at the previous steps.

```
1:  procedure BUILDQUERYFOR(example)                    ▷ Returns a String[]
2:      dependants ← getDependants(example)
3:      arguments ← getArguments(example)
4:      r ← ∅
5:      target classes ← union(dependants, arguments, example)
6:      for all t ∈ target classes do
7:          terms ← process(t)
8:          r ← r + terms
9:      end for
10:     return r
11: end procedure
```

Algorithm 4.2: Algorithm for building queries.

Regardless of what target classes are used or how they are combined, the output of the text mining process is always a collection of stems. Then, EasySOC uses the stems for building a vector $\vec{q} = (e_0, ..., e_n)$, where each element e_i represents the weight of a distinct stem for the component being outsourced as is explained in next section.

4.4 The partitioned Vector Space Model

Once relevant information has been gathered, the next step consists in representing it in a way that allows discoverers to efficiently retrieve proper services. The EasySOC approach adapts the Vector Space Model for generating spatial representations of gathered information. In order to map a collection of stems onto a vector $\vec{v}$, each different stem is represented as a dimension, in which the value, or weight, is given by a term weighting technique (Salton and Buckley, 1988).

Term Weight A quantity representing the importance of a term for a document or a collection of documents.

The EasySOC adapted VSM uses TF-IDF (term frequency-inverse document frequency), a weighting technique that combines two factors: TF and IDF. TF determines that a term is important for a document if it occurs often in it. On the other hand, terms which occur in many documents are rated as less important because of their IDF value. Formally, for each term t_i of a document d, $tfidf_i = tf_i \bullet idf_i$, with:

$$tf_i = \frac{n_i}{\sum_{j=1}^{T_d} n_j} \tag{4.1}$$

where the numerator (n_i) is the number of occurrences within d of the term being considered, and the denominator is the number of occurrences of all terms within d (T_d), and:

$$idf_i = \log \frac{|D|}{|\{d : t_i \epsilon d\}|} \tag{4.2}$$

where $|D|$ is the total number of documents in the corpus and $|\{d : t_i \epsilon d\}|$ is the number of documents where the term t_i appears.

The process of mapping collections of terms onto vectors using TF-IDF can be illustrated by means of a simple example. Let us suppose we want to find services for providing information about books covering a topic, and there are 2 services belonging to a category named "book" and 2 services belonging to a category named "movie". The corresponding TF-IDF-based vectors are as follows:

$$\begin{aligned}
\vec{v_0} &= (< book, 0.92 >, < searcher, 0.38 >) \\
\vec{v_1} &= (< book, 0.86 >, < searcher, 0.35 >, < topic, 0.35 >) \\
\vec{v_2} &= (< movi, 0.92 >, < topic, 0.38 >) \\
\vec{v_3} &= (< movi, 0.86 >, < searcher, 0.35 >, < topic, 0.35 >)
\end{aligned}$$

The main drawback of TF-IDF weighting technique is that it requires to recompute the idf_i factor when the collection is modified, i.e., idf_i is a collection-dependent factor. For instance, as long as new services are published in a registry, this factor must be updated. On the other hand, vectors representing queries do not impact on this factor, because they do not belong to the collection. Moreover, query vectors are intended to be temporal, thus these are discarded after retrieving related services. Despite of this shortcoming, current implementation of the EasySOC registry uses TF-IDF because this combined heuristic has shown to be suitable for weighting terms present in Web Service descriptions (Stroulia and Wang, 2005).

Now, let us suppose that the query *"book topic"* is generated, and in turn it is taken as input. Mapping the query onto the space shown above, generates a vector $\vec{q} = < book, 0.92 >, < topic, 0.38 >$. The next step of the discovery process, after generating the query vector $\vec{q}$, is to find similar vectors within such space in order to obtain Web Services relevant to the query. This requires to match $\vec{q}$ against the vector space in order to find nearest neighbours, which can be a compute intensive task when the number of services is large (Schmidt and Parashar, 2004). Thus, as part of the EasySOC approach a novel space reduction mechanism based on Rocchio's classification algorithm is presented (Joachims, 1997).

Rocchio's classifier suggests to partition a vector space into sub-spaces corresponding to a pre-established set of categories. Each sub-space is centered on an average vector, known as *centroid*, which stands for the documents that belong to the category associated with the sub-space. Formally, the centroid $\vec{c_i}$ for the documents that belong to category i is computed as:

$$\vec{c_i} = \alpha \frac{\sum_{\vec{d} \epsilon C_i} \vec{d}}{|C_i|} - \beta \frac{\sum_{\vec{d} \epsilon (D - C_i)} \vec{d}}{|D - C_i|} \tag{4.3}$$

with C_i being the sub-set of the documents from category i, and D the amount of documents of the entire data-set. First, both the vectors of C_i, i.e., the positive examples for a class, as well as those of $D - C_i$, i.e., the negative examples for a class, are summed up. The centroid vector is then calculated as a weighted difference of the positive and the negative examples. The parameters α and β adjust the relative impact of positive and negative training examples. Buckley et al. (1994) suggest to use $\alpha = 16$ and $\beta = 4$.

Returning to the discovery process, the vector $\vec{q}$ is compared against the centroids associated with all categories, in order to determine the category whose centroid maximizes vector similarity. Once a category has been selected, the query is compared *only* against the vectors that belong to this sub-space.

In essence, as will be further explained below, the EasySOC discovery algorithm is based upon a matching process that is composed of the following two steps:

1. determining the nearest category of the query,
2. comparing the query against the services belonging to the category returned by the previous step.

Nevertheless, n subsequent deduced sub-spaces are analyzed when no similar services have been found within previous sub-spaces in order to mitigate classification errors. Heß et al. (2004) refer to n as "n values of tolerance". Intuitively, this is more efficient

than matching $\vec{q}$ against the whole vector space. Besides, this reduces the number of dimensions of each individual sub-space, because services within an individual domain share the same sub-language (Losee, 1995). This will be formally analyzed in section 4.4.1. For the purposes of this thesis, a "sub-language" can be informally defined as a form of natural language used in a sufficiently restricted setting (Kittredge, 1982). Typically, a sub-language uses only a part of the language structures. For instance, in the business domain, words such as "economy", "competitive" and "currencies" occur often, while words such as "affine", "chebyshev" and "commutative" seldom appear.

The comparison between queries and either the centroids or the vectors of a selected partition, relies on a vector similarity measure. There are some different similarity calculations for finding related vectors (Korfhage, 1997). One measure that is widely used is the *cosine measure*, which has shown to be better than other similarity metrics in terms of retrieval effectiveness (Kim and Choi, 1999). The measure is derived from the cosine of the angle between two vectors. This assumes that two documents with a small angle between their vector representations are related to each other. As the angle between the vectors shortens, its cosine approaches to 1, i.e., the vectors are closer, meaning that the similarity of whatever is represented by the vectors increases. Formally:

$$cosineSimilarity(\vec{q}, \vec{s}) = \frac{\vec{q} \bullet \vec{s}}{|\vec{q}||\vec{s}|} = \frac{\sum_{i=1}^{T} e_{\vec{q},i} e_{\vec{s},i}}{\sqrt{\sum_{i=1}^{T} e_{\vec{q},i}^2 \sum_{i=1}^{T} e_{\vec{s},i}^2}} \tag{4.4}$$

The cosine similarity measure is used by the EasySOC discovery algorithm for matching a query $\vec{q}$ against each centroid or service $\vec{s}$, where $e_{[\vec{q}|\vec{s}],i}$ is their corresponding TF-IDF value in the i^{th} dimension. The denominator in equation 4.4 normalizes vectors $\vec{q}$ and $\vec{s}$. After normalization, a vector maintains its original direction, but its length becomes 1. This prevents a bias towards longer documents (Korfhage, 1997). Finally, results are sorted in decreasing order of cosine angles.

Having presented the space reduction mechanism and the vector similarity measure, lets return to the example presented in previous paragraphs to illustrate how they work together. In the example there are 4 vectors belonging to 2 categories: "book" and "movie". As the reader can note the vector space has 4-dimensions: "book", "searcher", "movi" and "topic". Then, under a one-step approach, we would perform perform 4 vector comparisons within a 4-dimensional space for finding vectors similar to the query $\vec{q}$. Instead, under the EasySOC two-steps approach, we perform 2 vector comparisons, in which the number of terms is 4 and, in turn, another 2 vector comparisons in which the number of

terms is 3. To do this, the centroids for each category must be calculated, being:

$$\begin{aligned} \vec{c}_{book} &= (< book, 0.93 >, < searcher, 0.34 >, < topic, 0.09 >) \\ \vec{c}_{movie} &= (< movi, 0.93 >, < topic, 0.34 >, < searcher, 0.09 >) \end{aligned}$$

Then, $\vec{q}$ is compared with the centroids (first step). Using cosine similarity the resulting similarities are:

$$\begin{aligned} cosineSimilarity(\vec{q}, \vec{c}_{book}) &= 0.898 \\ cosineSimilarity(\vec{q}, \vec{c}_{movie}) &= 0.130 \end{aligned}$$

The centroid associated with "book" category maximizes the similarity, therefore the query is compared against $\vec{v_0}$ and $\vec{v_1}$ (second step). As mentioned before, the first step requires 2 vector comparisons within a 4-dimensions space, and the second step requires another 2 vector comparisons within a 3-dimensions one. This is because the vectors within the sub-space named "book" contains the terms "book", "topic", and "searcher".

```
1:  procedure DISCOVER(example, tolerance)   ▷ Returns a list of candidate Web Services
2:      vector ← CREATEVECTOR(example)
3:      subspaces ← CLASSIFY(vector)
4:      candidates ← ∅
5:      i ← 0
6:      while i < tolerance do
7:          for all service ∈ subspaces[i] do
8:              if COSINESIMILARITY(vector, service) > Threshold then
9:                  INSERT(service, candidates)
10:             end if
11:         end for
12:         inc(i)
13:     end while
14:     return candidates
15: end procedure
```

Algorithm 4.3: Main steps of EasySOC approach to discover services.

To conclude, Algorithm 4.3 summarizes the different stages of the EasySOC approach to match queries onto service vectors. The algorithm receives as input a textual description representing an example of what a discoverer is looking for, and returns a list of candidate services. At step 2 the algorithm represents the example as a vector. At the 3^{th} step the algorithm classifies the example vector. The previous step returns a list of sub-spaces that has been ordered by taking into account the cosine similarity between the example and available centroids. Accordingly, the sub-space whose centroid is more similar to the

example is at the top of the list of sub-spaces. Then, from steps 6 to 13 the example is matched onto the services of the first n sub-spaces, where n is determined by *tolerance.* Tolerance indicates how many sub-spaces must be analyzed.

4.4.1 Time complexity analysis

The two steps of the EasySOC discovery algorithm rely on assessing cosine similarity measure. The computational complexity of assessing cosine similarity measure between two vectors takes linear time and depends on the number of dimensions of the vector space, i.e., the number of different terms T. Formally:

$$O(T) = \sum_{1}^{T} 2C_M + C_s \tag{4.5}$$

where C_M and C_S are two constants representing the corresponding cost of multiplication and square root operations, respectively. Accordingly, the complexity of the aforementioned first step is $O(CT)$ with C being the number of categories, and the second step takes $O(S_C T_C)$ with S_C being the maximum number of services per category and T_C being the maximum number of different terms per category. Therefore, the complexity of the complete discovery process of the EasySOC approach is:

$$O(max(CT, S_C T_C)) \tag{4.6}$$

The number of categories C is, at most, equal to the number of available services S, $C \leq S$, for example if each service belongs to a different category. Conversely, if there is only one category, $C = 1$, the maximum number of services per category S_C is equal to S, then $S_C \leq S$. Finally, the maximum number of different terms per category T_C is, at most, equal to T, $T_C \leq T$. On the other hand, the complexity of comparing a query against the whole vector space, i.e., a one-step approach, is $O(ST)$, thus under $C \leq S$, $S_C \leq S$ and $T_C \leq T$, a two-step matching process reduces the time complexity. However, if $C + S_C \geq S$, this is, $C = S$ or $S_C = S$, a two-step approach requires *only* one more vector comparison. To sum up, a two-step approach is more efficient than a one-step approach, otherwise if there is only one category or S categories, under these extreme conditions the former requires one more vector comparison.

As an example, Figure 4.7 illustrates the complexity of querying a registry with 8 available services divided in 2 categories. The first row shows the complexity of comparing a query against all vectors, which results in $O(ST)$. The second row shows the complexity of each individual step of our approach, which results in $O(max(CT, S_C T_C))$.

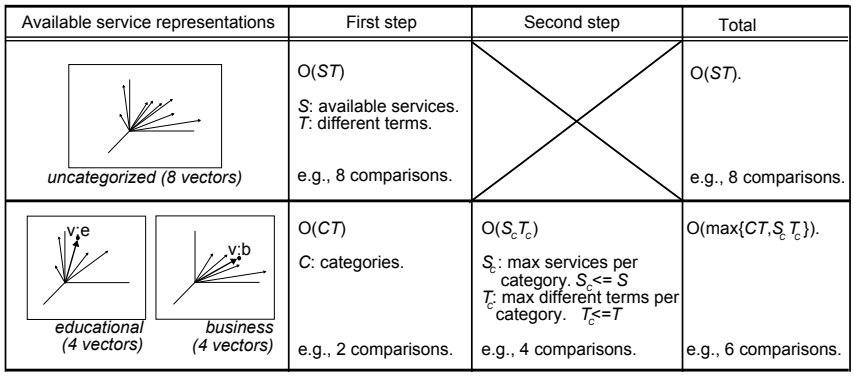

Figure 4.7: Time complexity example.

4.5 Web Service discoverability anti-patterns

Previous sections have shown that the proposed partitioned Vector Space Model enables to promptly retrieve Web Services. However, nothing was said with regard to the retrieval effectiveness of the proposed discovery process. With the EasySOC approach to represent services and queries as vectors, and with syntactic-based registries in general, looking for services related to a query heavily depends on how appropriate for discovery the WSDL documents and queries are. Therefore, the representativeness of names and comments within WSDL documents is crucial.

How explanatory service descriptions are, influences service chances of being selected by the human discoverer who is using a syntactic-based registry. This is because this kind of service registries returns a list of candidate WSDL documents, which the user who performs the discovery must analyze. In other words, while semantics-based registries are intended to automatize service discovery, with syntactic-based ones a human discoverer has the final word. The intelligibility of WSDL documents plays a very important role here. Below a novel catalog of discoverability anti-patterns to assist publishers in the creation of more discoverable services is presented as a main part of the EasySOC approach.

A body of WSDL documents that were gathered from Internet repositories by Heß et al. (2004), has been analyzed looking for recurrent bad practices that may deteriorate the retrieval effectiveness of syntactic-based registries. This data set consists of 391 WSDL documents. Initially, 130 files of the data set were manually revised and a detailed list of bad practices that frequently occur within this subset was documented.

Bad practice Is a frequent practice present in a body of public services that attempts to prevent the discovery and understanding of any service.

Table 4.4: Web Service discoverability anti-patterns.

Name	Concern	Problem	Manifest	Suggested solution
Inappropriate or lacking comments	Documentation	Occurs when: (1) a WSDL document has no comments, or (2) comments are inappropriate and not explanatory.	(1) Evident, or (2) Not immediately apparent	Create explanatory comments and place them in the correct part of the WSDL document.
Ambiguous names	Documentation	Occurs when ambiguous or meaningless names are used for denoting the main elements of a WSDL document.	Not immediately apparent	Replace ambiguous or meaningless names with representatives names.
Redundant port-types	Design	Occurs when different port-types offer the same set of operations.	Evident	Summarize redundant port-types into a new port-type.
Low cohesive operations in the same port-type	Design	Occurs when port-types have weak semantic cohesion.	Not immediately apparent	Divide non-cohesive operations into different port-types.
Enclosed data model	Representation	Occurs when the data-type definitions used for exchanging information are placed in WSDL documents rather than in separate XSD ones.	Evident	Move data-type definitions from WSDL documents to XSD files.
Redundant data models	Representation	Occurs when many data-types for representing the same objects of the problem domain coexist in a WSDL document.	Evident	Summarize redundant data-types into a new data-type.
Whatever types	Representation	Occurs when a special data-type is used for representing any object of the problem domain.	Evident	Replace Whatever types with data-types that properly represent the required objects.
Undercover fault information within standard messages	Design	Occurs when output messages are used to notify service errors.	Present in service implementation	Use fault messages for conveying error information.

Through this section, each identified bad practice is studied to provide a sound and practical solution. Table 4.4 presents a comprehensive list of the resulting anti-patterns using the following template:

Name A succinct name to convey the essence of the anti-pattern.

Concern A classification of the anti-pattern related to: problems on how a service inter-

face has been designed, problems on the comments and identifiers used to describe a service, and problems on how the data exchanged by a service are modeled. We refer to these three types of concerns as *Design*, *Documentation* and *Representation*, respectively. In the context of WSDL, the Design concern deals with the organization of port-types and operations. The Documentation concern deals with comments and names associated with port-types, operations and messages. Finally, the Representation concern deals with type definitions.

Problem The commonly occurring bad practice that relates to the anti-pattern.

Manifests Another classification based on how an anti-pattern manifests itself. An anti-pattern is *Evident* if it can be detected only by analyzing the structure, or syntax of the WSDL document. *Not immediately apparent* means that detecting the anti-pattern requires not only an syntactical analysis but also a semantic one. Finally, *Present in service implementation* anti-patterns may not show themselves in the WSDL document, thus requiring the execution of the associated service to be detected.

Suggested solution The refactored solution that solves the problem.

4.5.1 Inappropriate or lacking comments

Commonly, WSDL documents are not well documented (Fan and Kambhampati, 2005), or worse some WSDL documents are not documented at all. Placing explanatory comments within a WSDL file makes the intended functionality of its associated service easier to understand. Furthermore, syntactic service registries exploit this kind of documentation, present at WSDL documents, to support discovery.

When a WSDL document has no comments, it is said that it evidently suffers from Inappropriate or lacking comments anti-pattern. However, inappropriate comments may be not immediately apparent. A WSDL document is said to be well documented when each of its operations has a concise and explanatory comment, which describes the semantics of the offered functionality (Kramer, 1999). Moreover, as WSDL allows providers to comment each part of a service description separately, then a good practice is to place every <documentation> tag in the most restrictive ambit possible. For instance, if the comment refers to a specific message, it should be placed in that message and not in the operation that uses the message. Instead, a badly commented operation typically includes information that is not directly related to its functionality, e.g., details about either the authors or licenses of the service.

The solution to the Inappropriate or lacking comments anti-pattern is to comment the different parts of a WSDL document. The left side of Figure 4.8 depicts a non-commented WSDL document for a service that translates a given text from English to German, whereas

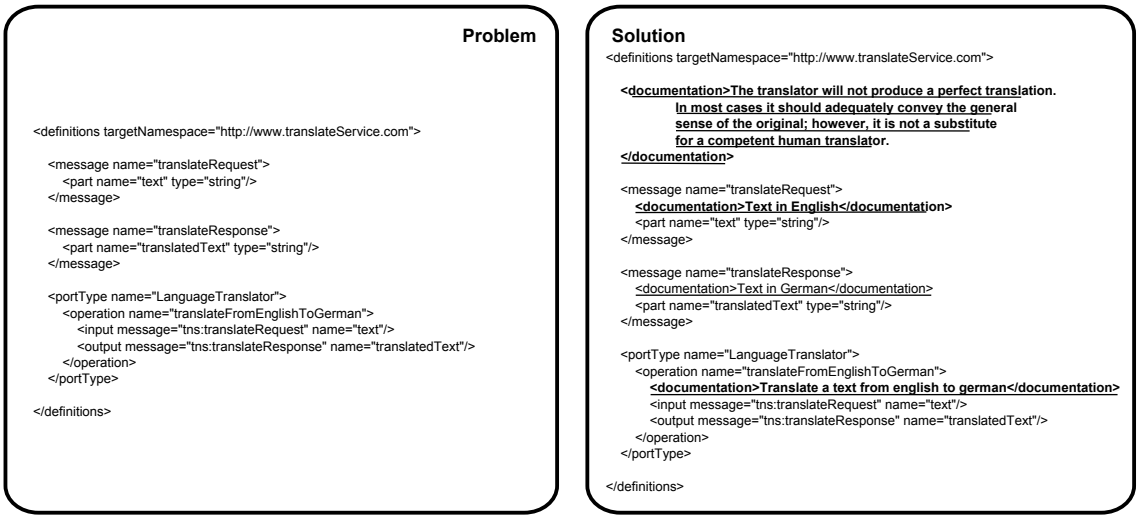

Figure 4.8: Example of Inappropriate or lacking comments anti-pattern.

the right side shows the same WSDL document after being improved. In Figure 4.8, each documentation element is underlined. As a result of the refactoring, the WSDL document not only is more intelligible, but also conveys more relevant terms, which is essential for syntactic registries such as (Dong et al., 2004), and many measures for assessing the similarity between Web Services (Kokash, 2006). With this example, specifically, by gathering terms from the documentation elements of the improved WSDL document, we obtained 4 occurrences of "translat" stem, 3 of "text" and 2 of both "english" and "german", respectively. This means that these comments allow gathering more terms related to the functionality of the service.

4.5.2 Ambiguous names

Another common bad practice is to use meaningless or cryptic terms to name port-types, operations, messages, and part elements. For example, in the Amazon AWSECommerce service[2], which is bound to SOAP over HTTP, all message parts are named "*body*". The name of a WSDL element is used not only as a unique identifier, but also as a descriptor. From a discoverer's point of view, a representative name should describe the semantics of an element, thus syntactic approaches to service discovery gather them.

Representative names should conform to some semantic and syntactic characteristics. Semantically, a representative name should describe what its element represent, then meaningless names, such as in0, arg1 or foo, should be avoided. Moreover, if there are two or more elements within a WSDL document standing for the same concept, these elements should be equally named. For instance, if an operation receives user's details

[2] `http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl`

as input and another operation produces user's details as output, their corresponding message parts should have the same name.

Syntactically, the name of an operation should be in the form: <verb> "+" <noun>, because an operation is an action. In the case of a message, a message part, or a data-type, its names should be a noun or a noun phrase, otherwise it may mean that a message conveys control information. This case is related to the Undercover fault information within standard messages anti-pattern, which will be described in section 4.5.8.

Moreover, as syntactic registries rely on popular naming conventions, such as JavaBeans or Hungarian notations, to split long names (Dong et al., 2004; Jian and Zhaohui, 2005; Kokash, 2006), if a name is composed by two or more words, the name should be written according to common notations. For example, the name "thisisthenameofanelement" should be rewritten as "thisIsTheNameOfAnElement" or "this_is_the_name_of_an_element". Besides, the latter names are clearly easier to be read than the original one. Another consideration is the length of a name. A name should be neither too short nor too long. A recommended length is between 3 and 15 characters (Blake and Nowlan, 2008).

The solution to the Ambiguous names anti-pattern is to replace meaningless names with names that follow the conventions mentioned before. Moreover, names in the refactored WSDL document should be consistent. It means that the names in the WSDL document and the concepts represented by these names should have an univocal correspondence. As a result, a refactored WSDL document is free from ambiguous names. Moreover, a refactored WSDL document allows syntactic registries to extract more terms related to its intended functionality and its problem domain.

4.5.3 Redundant port-types

Another common bad practice is to repeat port-types. A redundant port-type is one that consists of the same set of operations that are offered by another port-type of the same Web Service. Despite repeating the interface of a service might seem like a weird thing to do, developers typically define two or more port-types that offer the same operations with the same messages, but each port-type is bound to a particular transport protocol. . For example, the FraudLab service[3] defines the "same" port-type three times, but binds each one to a different protocol. Moreover, typically the names of redundant port-types start by describing the offered functionality, but each of them ends with a different abbreviation that represents a concrete transport technology, like "FindServiceSoap" of Microsoft Bing Maps platform services[4]. In the end, this springs unnecessarily big and

[3] `http://ws.fraudlabs.com/fraudlabswebservice.asmx?wsdl`
[4] `http://staging.mappoint.net/standard-30/mappoint.wsdl`

puzzling WSDL documents.

For example, the left side of Figure 4.9 depicts a service for translating a given English text into German. This translation service is defined as having two "different" port-types for consuming the service with either SOAP or HTTP, respectively. In this example, "LanguageTranslatorSOAP" and "LanguageTranslatorHTTP" port-types define the same set of operations. Indeed, they define the same messages.

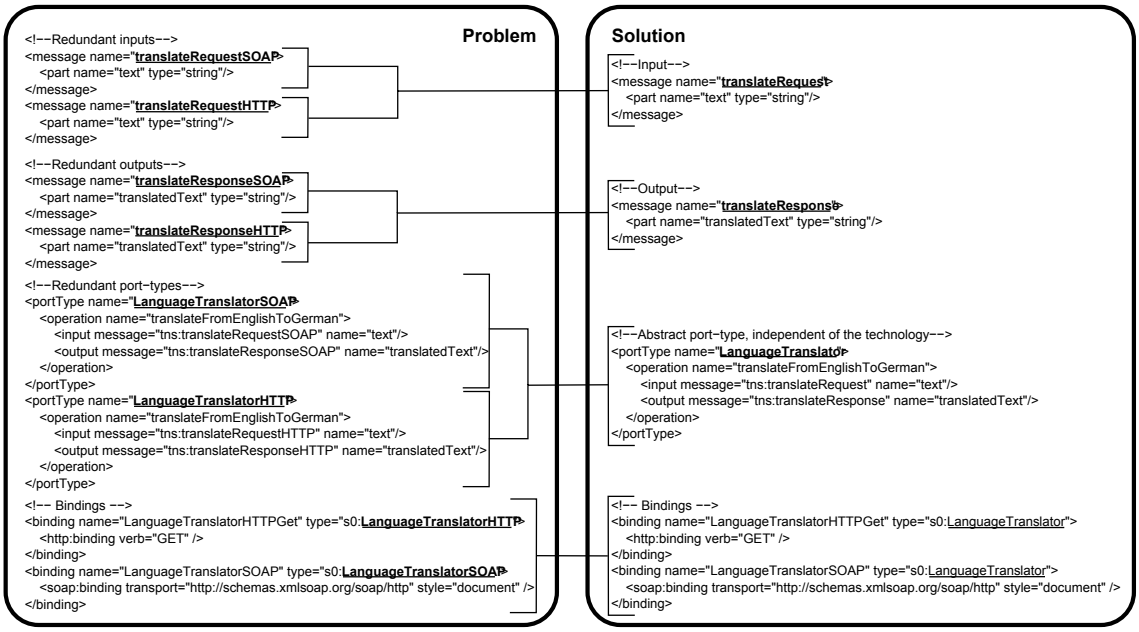

Figure 4.9: Example of Redundant port-types anti-pattern.

Redundant port-types may be considered as an attempt to influence the ranking of Web Services returned by a syntactic registry, apart from the obvious obstacle that this represents to isolate the potential consumers of a component from the implementation details (Yourdon and Constantine, 1979). Commonly, syntactic registries determine that a term is important for a WSDL document if it often occurs in that document, i.e., the higher the frequency of the term within a document is, the higher its importance becomes (Dong et al., 2004; Kokash, 2006). Based on the importance of gathered terms, syntactic registries rank Web Services similar to a given query. When publishing WSDL documents in a syntactic registry, if providers repeat the definition of the offered interfaces, the registry will gather more occurrences of relevant terms. As a result, Web Services with redundant port-types will have more chances of being ranked first. This situation is analogous to what has been described as Google Bombing[5].

The solution to the Redundant port-types anti-pattern is to combine redundant port-types into a single one that does not include technological aspects. The refactored code

[5]Google Bombing, `http://en.wikipedia.org/wiki/Google_bomb`

for the translation example is shown in the right side of Figure 4.9, in which solid lines emphasize the relations between the elements of the original WSDL document and their refactored counterparts. Basically, the refactoring consists in removing redundant port-types and messages, keeping one of each kind, naming them with protocol-independent names and updating the definitions of the concrete bindings. Accordingly, if the refactored WSDL document is free from redundant code, then a syntactic registry will not extract redundant terms from it. Besides, the improved WSDL document will be conciser than the original one.

4.5.4 Low cohesive operations in the same port-type

Another commonly found bad practice is to place semantically unrelated operations in a unique port-type, although modules with high cohesion tend to be preferable in structured design. Cohesion is the degree of the strength of functional relatedness of operations within a service. The operations in the services must be highly related to one another, i.e. highly cohesive (Papazoglou and Heuvel, 2006). For example, the Amazon EC2 service[6] has 74 operations for managing images, volumes, security, instances, snapshots, among others, grouped in a port-type. By grouping cohesive operations within separate port-types, i.e., a port-type for managing images, another for volumes, each port-type may be more cohesive, while avoiding problems similar to "God classes".

Weak cohesion often occurs in Web Services that place operations for checking the availability of the service and operations related to its main functionality into a single port-type. An example of this bad practice is to include operations such as "isAlive", "getVersion" and "ping" in a port-type, though the port-type has been designed for providing operations of a particular problem domain, e.g., to give exact information on commodities. From the perspective of syntactic approaches to service discovery, WSDL documents with highly-cohesive port-types convey terms that are representative of the domain of their container services mostly. Then, when asking a syntactic registry using queries comprising terms related to the domain of the services, these services will have higher probabilities of being discovered.

The solution to the Low cohesive operations in the same port-type anti-pattern is to remove the non-cohesive operations from their common port-type and put them into either a new port-type or a new Web Service. The idea is to keep only related operations within the original port-type, to increase its cohesion. At this point, if non-cohesive operations are defined in a new port-type, the Low cohesive operations in the same port-type anti-pattern will occur in the new port-type. Therefore it may be required to iterate the proposed solution until neither the original port-types nor the new port-types contain

[6] `http://s3.amazonaws.com/ec2-downloads/2009-10-31.ec2.wsdl`

non-cohesive operations. In the end, a refactored WSDL document contains more port-types, but they are more cohesive than the port-types in the original WSDL document.

4.5.5 Enclosed data model

Another detected bad practice is to confine ad hoc data model definitions in each service description that requires them. An enclosed data model is a data model that is placed within a WSDL document, and can be used *only* from the operations described in their container WSDL document. Instead, an imported data model is developed in isolation from a service description, placed within a separate XSD document, and referenced from WSDL documents as required. With respect to Web Service discoverability, we assume that developers use best practices for naming and modeling data-types, when building their data models within separate XSD files. This assumption is usually true, because imported data models are written to be reused by other developers. As syntactic registries extract relevant terms from data-types associated with messages (Jian and Zhaohui, 2005; Wang and Stroulia, 2003), data models conceived for being reused may positively impact the precision of this kind of registries.

Imagine a Web Service for checking stocks when the stock market is open and another service for checking stocks when the stock market is closed. The two services offer an operation that retrieves market information, the only difference is the moment when that information is collected. Suppose that the data-types of one service are modeled following an enclosed approach, thus the corresponding WSDL document contains a data-type definition named "StockQuote" that stands for market information. In the same way, the developers of the other service define a similar data-type, but named "StockInfo", within their WSDL document. Although the data-types "StockQuote" and "StockInfo" represent the same concept, their definition code is not reused. However, if we keep only one of these data-types, improve the names of its attributes and comments, place it in a separate schema document, and in turn, reference it from both WSDL documents, then the data model is reused.

The solution to the Enclosed data model anti-pattern is to use best practices for naming and modeling data-types, defining them in separate XSD documents and, in turn, combining separate model definitions as required using the tags "include" or "import". This allows publishers to share and reuse data models rather than re-design ad hoc data-types for each new Web Service. Additionally, reusing data-models ensures that some ambiguous names (see section 4.5.2) will be avoided, which states that element names should be consistent. It is worth noting that when data-types are not going to be reused or are very simple, they can be part of the WSDL document to make it "self-contained", but they should be designed following best practices for naming and modeling data-types.

4.5.6 Redundant data models

Defining the same data-type two or more times is another common bad practice. Suppose a developer combines the "enclosed versions" of the Web Services for checking stocks at days' open and at days' close into the same WSDL document. Then, two data-types for representing the same object of the problem domain, "StockQuote" and "StockInfo" will coexist in the WSDL file. In general, a redundant data model occurs when, at least, two data-type definitions stand for the same exchangeable information in a WSDL document. For a syntactic registry, redundant data models may produce the same effect as redundant port-types. Besides, this anti-pattern, like the Redundant port-types anti-pattern, causes unnecessarily big and puzzling WSDL documents from the perspective of a human discoverer.

The Redundant data models anti-pattern is a problem often caused by a bad representation of the required data-types. The refactoring strategy to correct this anti-pattern consists in replacing redundant data-types with a single data-type, and changing references to the old data-types in message parts for references to the refactored one. In consequence, the refactored version of the service description is free from redundant XSD code. Then, the service definition may be conciser and easier to be understood by third-party discoverers than its original version. Optionally, the refactored data model should be defined in a separate XSD document to prevent occurrences of the Enclosed data model anti-pattern.

4.5.7 Whatever types

Another common practice found in WSDL documents is to use general purpose data-types for representing any object of a problem domain. We refer to this kind of data-types as Whatever types. Typically, a Whatever type relies on the use of the XSD constructs xsd:any and xsd:anyAttribute. Below we present the XSD code for defining a data-type, called DataSet, which developers frequently use for exchanging a sequence of almost any complex XML structure between providers and consumers:

```
<xsd:element name="DataSet" nillable="true">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="s:schema"/>
         <xsd:any/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

A DataSet instance can be any valid XML, even an empty XML. While this might be seen as an alternative for building extensible operations (Pasley, 2006), it obscures the domain

and range definitions of the operations that convey this kind of types in their messages. In consequence, this data model design anomaly makes operations hard to be understood by third-party discoverers.

As an example, suppose a port-type named "RandomGenerator" comprising one operation whose signature is "generate():DataSet". The signature and data-type of this operation are ambiguous. A human discoverer cannot determine what the structure of the operation response will be like until they invokes the operation, and neither whether he/she will receive more than one object. In general, when using Whatever types there is no way to express which extensions are supported (Pasley, 2006). Moreover, this kind of types typically does not carry explanatory terms, thus hindering discoverability through syntactic registries.

Instead, suppose another operation, whose signature is "generate():RandomNumber" with RandomNumber being:

```
<xsd:simpleType name="RandomNumber">
  <xsd:restriction base="xsd:double">
    <xsd:minInclusive value="0"/>
    <xsd:maxExclusive value="1"/>
  </xsd:restriction>
</xsd:simpleType>
```

From the signature of this new operation it may be derived that the operation returns one random number. Moreover, as RandomNumber extends the double primitive data-type and limits its range between 0 and 1, a human discoverer may derive that the operation returns a random number represented as a double between 0 and 1.

The Whatever types anti-pattern can be symptomatic from bad data model designs or the desire to make Web Service interfaces extensible (Pasley, 2006). When the first problem occurs, the solution associated with this anti-pattern comprises replacing all "Whatever" definitions with proper data-types and updating references to them. A proper data-type should be intended to merely convey either the information that an operation needs as input or produces as output, and should be named with a representative name. In the end, the refactored operation and its message data-types will convey relevant terms rather than "DataSet". Instead, if this anti-pattern stemmed from extensible interfaces, the solution is to apply the versioning strategy proposed in (Pasley, 2006). Broadly, this versioning strategy recommends to build new backward compatible data-types by using the extension mechanism of XSD, which allows developers to define a new data-type from one created previously.

4.5.8 Undercover fault information within standard messages

Using output messages to inform about errors that occur during the execution of an operation is a common bad practice. In consequence, an output message is designed for exchanging error information when the operation fails, or a result value when the operation successfully finishes its execution. Clearly, to do this the output message of an operation must be defined using a flexible data-type. This, besides causing occurrences of the Whatever types anti-pattern, obscures the underlying logic of an operation. Returning fault information within output messages may negatively impact syntactic registries that exploit either the names or the XML structure of message parts, such as (Dong et al., 2004; Jian and Zhaohui, 2005) and (Stroulia and Wang, 2005), respectively.

This anti-pattern normally appears when a message is associated with a general purpose data-type, e.g., the DataSet type discussed previously. A clear example of this bad practice takes place in the Amazon SimpleDB service[7]. It offers an operation named Select whose Output message carries a set of items when everything goes well, otherwise it carries error information.

Another frequent form of this anti-pattern requires a message definition of three parts: a part informs whether an error occurred, another part conveys error details if any, and the third part carries the operation result. For example, following this "three-parts" approach the output message of an operation for calculating the factorial of a non-negative integer given as input is:

```
<message name="calculateFactorialResponse">
    <part name="isError" type="xsd:bool"/>
    <part name="stackTrace" type="xsd:string"/>
    <part name="factorial" type="xsd:long"/>
</message>
```

The part "isError" communicates whether an error occurred. If an error occurs, then the part "stackTrace" will convey its description, otherwise it will convey no data. On the other hand, if there is an error, "factorial" will be empty, if not it will transport the calculation result. Furthermore, now the output message not only transports application data, but also tells the client what to do. This is an special case of control coupling, a situation that should be avoided in structured design (Yourdon and Constantine, 1979). From the perspective of syntactic approaches to service discovery, the presented control parameter conveys the terms: "error stack trace factorial", in which 75% of them are unrelated to the factorial calculations domain.

The solution to the Undercover fault information within standard messages anti-pattern relies on WSDL support for handling error information. Then, to solve this anti-pattern,

[7] http://sdb.amazonaws.com/doc/2009-04-15/AmazonSimpleDB.wsdl

error information should be exchanged within fault messages. Returning to the factorial calculator example, the refactored code is:

```
<message name="calculateFactorialFault">
    <part name="errorDescription" type="xsd:string"/>
</message>
<portType name="FactorialCalculator">
    <operation name="calculateFactorial">
        <documentation>Returns the factorial calculator
              for a given number</documentation>
        <input message="tns:calculateFactorialRequest" name="number"/>
        <output message="tns:calculateFactorialResponse" name="result"/>
        <fault message="tns:calculateFactorialFault" name="errorDetails"/>
    </operation>
</portType>
```

The refactored operation has three messages: input, output and fault messages. Now, input and output messages only exchange application data between client and server. Additionally, the fault message has been specially designed for supplying invokers with a description of errors. In this example, as error information is a textual description, the fault is associated with a string. In other contexts, it could be associated with a complex-type for representing the stack-trace of the service operation. Finally, the refactored WSDL document is free from general purpose types and control coupling. Another good example of how to deal with operation errors is shown in the services of the NASA ECHO project, in which all offered operations[8] use Fault messages to transport different kinds of errors, such as "InvalidArgumentFault", "ItemNotFoundFault" and "AuthorizationFault".

4.5.9 Employing the catalog of discoverability anti-patterns

Given a WSDL document the anti-patterns described through section 4.5 can be applied in any order. However, it is recommendable to apply the remedies according to the following six steps:

1. Separating the schema from the definition of the offered operations.
2. Removing repeated WSDL and XSD code.
3. Putting error information within Fault messages and only conveying operation result within Output ones.
4. Replacing WSDL element names, with explanatory names if they are cryptic.

[8] http://api.echo.nasa.gov/echo-wsdl/v10/ExtendedServicesService.wsdl

5. Drawing non-cohesive operations from their port-types and put them into a new port-type.

6. Documenting the operations.

The first step means to put complex data-type definitions into a separated schema file. By doing so, these data definitions will better suited for reuse and the resulting service contract will be smaller and easier to read than its original version. However, when data-types are not going to be reused or are very simple, they can be part of the WSDL document to make it "self-contained".

The second step deals with redundant code in both the WSDL document and the schema. Repeated WSDL code might stem from port-types tied to a specific protocol, whereas redundant XSD from data definitions bounded to a particular operation. Therefore, repeated WSDL code can be removed by defining a protocol independent port-type. Similarly, to eliminate redundant XSD code, repeated data-types should be abstracted into a single one, and message part references changed for references to it. As the remedies associated with Redundant port-types and Redundant data models anti-patterns eliminate redundant elements, the anti-patterns that are conveyed in these elements will be eliminated as well, i.e., the Ambiguous names anti-pattern.

The third step intends to separate error information from output information. To do this, error information should be removed from Output messages and placed on Fault ones. Moreover, as many Fault massages should be defined as different operation errors exist.

The fourth step aims to improve the representativeness of WSDL element names by renaming non-explanatory ones. Grammatically, the name of an operation should be in the form: <verb> "+" <noun>, because an operation is an action. While, message, message part or data-type names should be a noun or a noun phrase because they represent the objects on which the operation execute. If those names represent actions it is possible that the information conveyed in those messages modify the operation behavior. Additionally, the names should be written according to common notations, and their length should be between 3 and 15 characters (Blake and Nowlan, 2008) because its facilitates both automatic analyzes and human reading, respectively.

The fifth step is to place operations in different port-types based in their cohesion. To do this, the original port-type should be divided into smaller and more cohesive port-types. This step should be repeated while the new port-types are not cohesive enough.

Finally, all operations must be well documented. An operation is said to be well documented when it has a concise and explanatory comment, which describes the semantics of the offered functionality. Moreover, as WSDL allows developers to comment each part of a service description separately, then a good practice is to place every <documenta-

tion> tag in the most restrictive ambit possible. For instance, if the comment refers to a specific operation, it should be placed in that operation.

It is worth noting that except for steps 4 and 6, the other steps may require to modify service implementations. Moreover, as a result of applying this guide, there will be two versions of a revised service description. Despite being out of the scope of this thesis, some version support technique is necessary to allow clients that use the old service version to continue using the service until they migrate to the new version (Juric et al., 2009).

4.6 Conclusions

This chapter described EasySOC, a new approach to ease the development of service-oriented applications, being mainly focused on publication and discovery tasks. EasySOC approach has been conceived to cope with the subset of problems related to service discovery that has been described in section 3.7.

First, as the spirit of this novel approach is to exploit the information conveyed within ordinary WSDL documents, UDDI entries, and client-side applications, without requiring all the specifications of full semantic techniques, it is expected that it can be transparently adopted. Second, EasySOC promotes the efficient usage of the current meta-data that support Web Service discovery by providing publishers with service classification assistance and a catalog of discoverability anti-patterns. Third, EasySOC contributes with novel methods to efficiently describe discoverers' intentions. Fourth, EasySOC presents a way to automatically reduce the search space during discovery, which may allow managing an ever increasing number of published services and facing the curse of success metaphor.

Central to EasySOC, is to assist publishers and discoverers on making better service descriptions and queries respectively, which directly impacts on the retrieval effectiveness of the proposed discovery approach. Therefore, it is possible to assume that, despite the lightweightness of EasySOC with respect to semantics-based approaches, it will achieve good precision for discovering relevant services, i.e., without the necessity for sacrificing none of the values of the "retrieval effectiveness versus costs of adoption" trade-off.

Chapter 5

Experimental Results

EasySOC is a novel approach to ease the development of service-oriented applications being mainly focuses on: help publishers to make more discoverable services, and allow discoverers to retrieve proper services without requiring heavy query descriptions nor imposing execution time overheads to the discovery process. In the previous chapter, details of the EasySOC approach were presented. This chapter describes the experiments that were carried out to provide empirical evidence about the practical soundness of the approach.

To conduct the experiments, an implementation of the EasySOC approach has been developed. Current implementation consists of two main components. One main component is the EasySOC registry, which is mainly in charge of gathering relevant information from Web Service descriptions, mapping gathered data onto the Vector Space Model, and attending discoverers' requests. The EasySOC registry has been implemented in Java. Another component is in charge of assisting discoverers to build their queries by gathering relevant information from their service-oriented applications. The latter component is called the EasySOC plug-in, since it is implemented as a plug-in for the Eclipse IDE.

The experiments can be organized in four groups according to which aspect of EasySOC approach is under evaluation. Specifically, one group concerns EasySOC search space reduction mechanism. Other group is related to the retrieval effectiveness of EasySOC registry. The third group of experiments is associated with the discoverability anti-patterns, and their impact on both discovery and users' ability to understand Web Service descriptions. Finally, another group of experiments is intended to evaluate the response time and memory usage of current implementation of the EasySOC approach.

The chapter is organized as follows. The next section describes the first group of experiments. Experiments for evaluating the retrieval effectiveness of EasySOC registry are reported in section 5.2. Section 5.3 describes anti-patterns-related experiments. The

evaluation of EasySOC implementation is shown in section 5.4. Accompanying each experiment description, its setting is detailed explained and results are discussed. Finally, section 5.5 presents the conclusions of the chapter.

5.1 Evaluation of EasySOC space reduction support

This section describes the experimental evaluation of the EasySOC support for reducing the search space. Two experiments for evaluating the space reduction support were performed. As explained in section 4.4, such support is based on an algorithm for classifying documents, which was conceived by J. Rocchio. First, Rocchio's algorithm was compared with two classic document classification algorithms, namely K-NN, and Naïve Bayes (Lewis, 1998). Second, classification results achieved using Rocchio's algorithm were compared with the results reported in (Heß et al., 2004), in which the authors combine Naïve Bayes and Support Vector Machine algorithms to classify WSDL files.

For the first experiment K-NN and Naïve Bayes algorithm implementations provided by the Rainbow toolkit (McCallum, 1996) and Weka framework (Frank et al., 2005) were employed. Then, a methodology for evaluating document classification algorithms called Repeated Holdout was followed. This methodology requires a data set of classified documents as input, WSDL documents in this particular case. Then, documents are chosen randomly from the data set to form the validation set, and the remaining items are retained as the training set. The training set is employed to build the classification system, or classifier. Once the classifier is built, it is employed to classify each validation data keeping record of whether it is correctly classified or not.

The accuracy of a classifier under evaluation is calculated dividing the number of correctly classified items by the size of validation data set. For example, if there are 10 validation data, and 5 of them are correctly classified, the accuracy obtained following the Holdout method is $\frac{5}{10} = 50\%$. In the Repeated mode, results from successive tests with different training sets, but of the same size, are averaged. Average accuracy is referred to the average of successive test results.

The data set employed for the experiment consists of 391 WSDL documents that were gathered from Internet repositories by Heß et al. (2004). This body of WSDL documents is classified in 11 categories. One thousand distributions of training and validation data sets were randomly generated. The random distributions were generated using 90% of the 391 documents as training data.

Table 5.1 shows the average accuracy for K-NN, Naïve Bayes, and Rocchio algorithms. As the reader can see, Rocchio's algorithm surpassed the other two, at least with the employed data set.

Table 5.1: Comparison between different classifiers.

Classifier	Average Accuracy
K-NN	39.59%
Naïve Bayes	79.38%
Rocchio	85.08%

In order to compare Rocchio's algorithm with a related work that combines Naïve Bayes and Support Vector Machine algorithms, the experiment reported in (Heß et al., 2004) was reproduced, by cloning its setting. In this sense, the same data set and evaluation methodology were used. With regard to the data set, the related work was evaluated with the aforementioned body of 391 Web Service descriptions. The employed evaluation methodology was Leave-one-out. With this methodology, one item is arbitrarily chosen from the initial data set to form the validation data and the remaining items are retained as the training data. Then, the isolated item is used to test the classifier, accounting whether it is correctly classified or not. Moreover, Heß et al. (2004) take into account a tolerance value for computing accuracy. A tolerance value of t represents that the correct classification is included in a sequence of $t + 1$ suggestions. Then, the authors averaged the accuracy over the 391 services for each tolerance value.

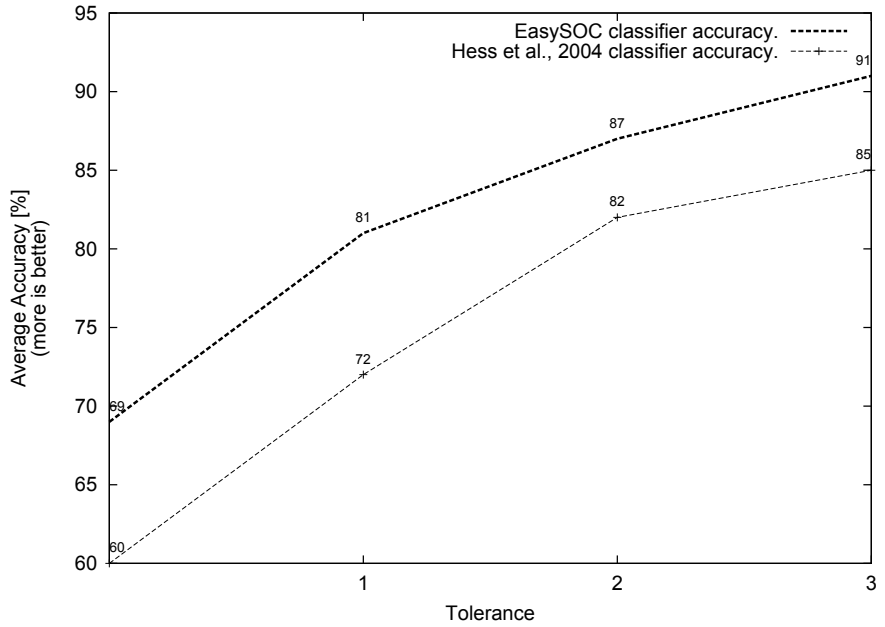

Figure 5.1: Comparison of the classification support.

The aforementioned experiment was reproduced using the Rocchio's classifier provided by EasySOC. Figure 5.1 depicts the results achieved by EasySOC and those reported by Heß et al. 2004. As shown in Figure 5.1, the averaged accuracy of our classifier surpassed that reported in (Heß et al., 2004) by at least 9% with two values of tolerance: $t = 0$

and $t = 1$.

The rest of this chapter presents experiments related to the EasySOC approach to discovery, mostly. As the results achieved using Rocchio's algorithm surpassed those achieved by K-NN, Naïve Bayes, and a combination of Naïve Bayes and Support Vector Machine algorithms, Rocchio's algorithm was used as the space reduction mechanism of EasySOC for the remaining experiments.

5.2 Evaluation of EasySOC approach to service discovery

This section describes the experimental evaluation of EasySOC approach to service discovery. As explained through chapter 4, the approach frees discoverers from writing queries by gathering relevant information from their service-oriented applications. Then, queries are represented as vectors and matched onto vectors standing for available services using our partitioned Vector Space Model. Finally, a list of candidates is returned to the discoverer who analyzes retrieved candidates. As supported by different experiments, users tend to select higher ranked search results (Agichtein et al., 2006), EasySOC registry was evaluated in terms of the proportion of relevant services in the retrieved list and their positions relative to non-relevant ones.

In general, the employed methodology for the next experiments consisted in supplying the EasySOC registry with the aforementioned collection of categorized Web Services, which have been gathered by Heß et al. (2004), and in turn querying it using different alternatives to query generation. For each particular query the Normalized Recall, Recall-at-n, and Precision-at-n measures were computed. Finally, computed measures were averaged over the total number of queries. For some experiments this methodology was slightly modified. Each particular case will be explained in detail. Below, the evaluation set, employed metrics, conducted experiments, and their results, are presented.

5.2.1 Evaluation set

To conduct the tests several service-oriented applications were built by outsourcing 30 Web Services, from which 150 queries were generated using EasySOC query expansion support. Each query was the result of supplying the text mining process, which is described in section 4.3, with specific source code files. Some of them represented a programmatic description of an external component being outsourced, while others represented the context in which it was used. Descriptions of such external components stand for "examples". Both the header and the methods of each example included proper comments in the associated source code file. For those methods that accepted user-defined data-types

as arguments, their associated source code files were properly commented as well. Another source code file represented an internal component that depended on the example. Likewise, both the header and the methods of each dependant included comments.

Different parts of the source code associated with each example, were combined to derive five query expansion alternatives per example. In the rest of this chapter we will refer to these query expansion alternatives as "Interface", "Documentation", "Arguments", "Dependants", and "All".

"Interface" alternative represents when gathering relevant information from an example programmatic functional description. With "Interface" alternative, the name of a component along with the names of its operations are only considered. Natural language descriptions, i.e., Javadoc comments, in the queries were not take into account. In fact, by omitting the second activity of the aforementioned text mining process the focus was on measuring the performance of the discovery mechanism using only mandatory pieces of information. Instead, the alternative named "Documentation" represents when gathering relevant information from the offered API and Javadoc comments of an example.

The combination of an example source code file and those files belonging to the list of arguments is called "Arguments". Alternatively, when combining an example source code file with its dependant source code file, we named it "Dependants". Finally, we refer to gathering relevant information from all available source code files as "All".

Table 5.2: Number of different extracted stems per query.

1. (4,7,19,19,30)	6. (4,7,7,17,17)	11. (4,6,30,11,34)	16. (3,7,17,32,40)	21. (5,9,10,19,20)	26. (3,9,11,21,23)
2. (3,5,5,14,14)	7. (4,8,12,26,30)	12. (1,5,5,10,10)	17. (5,8,17,21,29)	22. (3,10,19,20,27)	27. (5,8,36,31,59)
3. (4,8,11,16,19)	8. (3,6,12,10,16)	13. (4,7,22,11,24)	18. (5,8,12,26,28)	23. (3,7,9,16,18)	28. (4,10,16,18,24)
4. (4,8,14,20,26)	9. (4,6,30,11,34)	14. (4,8,8,20,20)	19. (2,7,14,30,34)	24. (4,7,9,15,17)	29. (5,10,12,25,27)
5. (5,15,15,20,20)	10. (4,8,16,22,29)	15. (2,5,7,15,17)	20. (4,7,15,20,28)	25. (4,7,11,17,20)	30. (6,15,29,31,43)

Table 5.2 summarizes each resulting query expansion alternative as a five-tuple:

$$query_i = (Interface_i, Documentation_i, Arguments_i, Dependants_i, All_i)$$

in which an element stands for the number of stems that resulted from supplying the EasySOC text mining process with the source code files associated with any of the described combinations. For instance, the 7^{th} query has 4 stems when processing its "Interface". On the other hand, the query has 8 different stems when processing its corresponding "Documentation" alternative. Adding terms from the descriptions of its operation arguments, i.e. its "Arguments" alternative, results in 12 stems, while 26 stems when using its corresponding "Dependants". Finally, when feeding the text mining process with the interface and documentation of the example, arguments, and dependants

classes, the number of different stems is 30. Therefore, the query five-tuple representation is $(4,8,12,26,30)$. Note that these results are coherent with the discussion accompanying the example shown in Table 4.3, and show that query expansion commonly augments the number of different stems of each query.

5.2.2 Metrics

The Normalized Recall (*NR*) is one of the most popular measures for evaluating and comparing information retrieval systems, because it returns one single number in contrast to paired recall-precision measure (Bollmann, 1983). Recall is a measure of how well an engine performs in finding relevant documents (Korfhage, 1997). Normalized Recall takes into account Recall and the position of each relevant retrieved document within the result list. Formally, the *NR* of a query with R relevant services is:

$$NR = 1 - \frac{\sum_{i=1}^{R} r_i - \sum_{i=1}^{R} i}{R(N-R)} \tag{5.1}$$

where N is the total number of retrieved services and r_i is the ranking within the result list of the i^{th} relevant service.

Precision-at-n computes precision at different cut-off points of the results list (Korfhage, 1997). For example, if the top 10 documents are all relevant to a query and the next 10 are all non-relevant, we have a precision of 100% at a cut-off of 10 documents but a precision of 50% at a cut-off of 20 documents. Formally:

$$Precision\,at\,n = \frac{RetRel_n}{n} \tag{5.2}$$

with $RetRel_n$ being the number of relevant documents retrieved at the n^{th} position of the result list. Two particular cases of this metric, a case for $n = 1$ and another for $n = R$, were used. The first case, Precision-at-*1*, indicates whether the first retrieved element is relevant or not. The second case computes the precision at the R^{th} position in the rank. The Precision-at-*R* particular case is known as *R*-Precision. Therefore, we will refer to Precision-at-*R* as *R*-Precision from now on. For example, if there are 10 documents relevant to the query within a data-set and they are retrieved before the 11^{th} position, we have an *R*-Precision of 100%, but if 5 of them are retrieved after the top 10 we have 50%.

As we mentioned, Recall computes how effective a discovery system is in retrieving relevant documents. Recall is 100% when every relevant document of a data-set is retrieved.

Formally:

$$Recall\,at\,n = \frac{RetRel_n}{R} \tag{5.3}$$

By blindly returning all documents in the collection for every query, i.e., n=collection size, we could achieve the highest possible recall, but looking for relevant services in the entire collection is clearly a slow task. We want to achieve good Recall in a window of *only* 10 retrieved services. We have chosen this window size because we want a good balance between the number of candidates and the number of relevant candidates retrieved. Moreover, we believe that a developer can easily examine 10 Web Service descriptions. Therefore, by setting $n = 10$, we refer to the actual number of relevant services up to only 10 candidates in the result list.

As some of the employed metrics require to exactly know the set of all services in the collection that are relevant to a given query, the data set has been exhaustively analyzed. The criterion employed to judge whether a WSDL document was actually relevant to a query states that: "if the operations of a WSDL document fulfilled the expectations previously specified in the example Java code, then a hit was produced". For example, if the developer required a Web Service for converting from Euros to Dollars, then a retrieved Web Service for converting from Yens to Dollars was considered non-relevant, even though these services were strongly related. Under this definition of "hit", only Web Services for converting from Euros to Dollars were relevant.

Once the whole data set was analyzed looking for WSDL documents that were relevant to the queries of the evaluation set, two peculiarities were found. First, 10 queries had associated only one relevant service. Second, for any query there were, at most, 8 relevant services. This severely impacts on Precision-at-n measures since if the relevant service is not ranked first for these 10 queries, then the corresponding measures will be 0%. Note these peculiarities and the definition of hit make the validation of the discovery mechanism very strict.

5.2.3 Comparison of term weighting techniques

Having presented the employed data set, evaluation set, and metrics, three experiments are described now. The first experiment was conducted to compare TF, and TF-ICF, with TF-IDF, because the former two weighting techniques do not dependent on the entire collection, as the latter does. As described in Definition 4.1, mapping a new document onto a vector using TF requires to know the term frequency distribution of that document only. Term Frequency-Inverse Corpus Frequency (TF-ICF) (Reed et al., 2006) is a novel weighting technique that requires to know the term frequency distribution of a smaller data set to approximate that of a larger data set. The TF-ICF approach assumes that

the document frequencies of terms in a specific domain of English usage follow some distribution (Reed et al., 2006). Note that the idea of TF-ICF is similar to the use of training data in a classification context. Instead, in order to use TF-IDF one needs to know the number of documents of the entire data set in which a term occurred at least once.

In order to conduct this experiment TF, and TF-ICF support have been added to the EasySOC registry implementation. Then, three EasySOC registries, namely TF, TF-ICF and TF-IDF, were deployed. The employed data set was published in the TF and TF-IDF registries. On the other hand, 50 distributions of 100 WSDL documents were randomly generated to be used as training set, and successively published them in the TF-ICF registry to evaluate it using different training data.

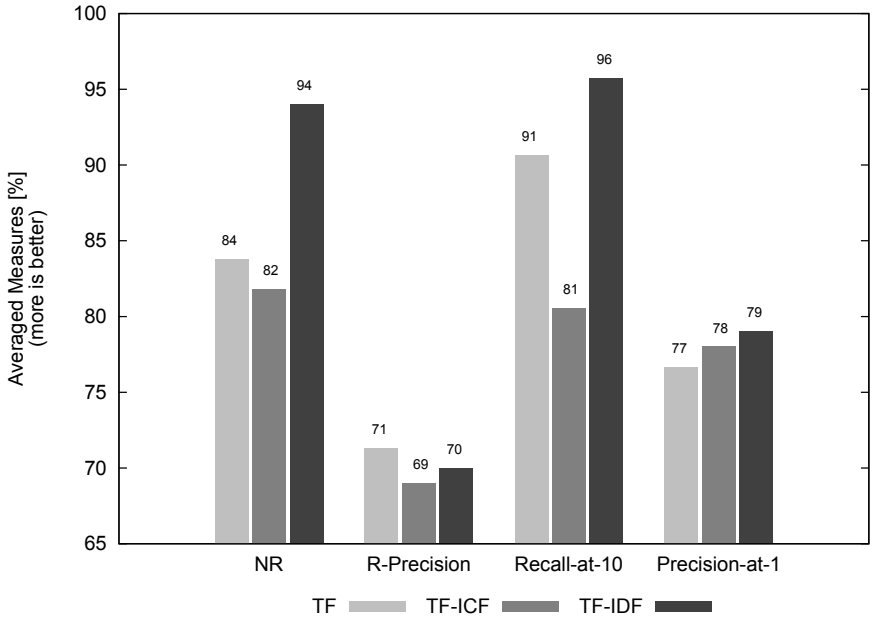

Figure 5.2: Comparison of TF, TF-ICF and TF-IDF term weighting techniques.

As described in previous section, the evaluation set consists of 150 queries, i.e. 30 queries per query expansion alternative. Then, the employed metrics were assessed for each query expansion alternative by averaging the metric results over its 30 queries. When using the TF-ICF registry, this methodology was repeated in order to calculate the employed metrics using the 50 training set distributions, average the results over 50, and calculate standard deviation.

Figure 5.2 shows the results of the first experiment. In Figure 5.2, bars represent the average value achieved for each analyzed metric using different query expansion alternatives. For example, using TF-IDF the achieved NR results were 92.77%, 94.18%, 94.85%, 93.58%, and 94.32%, for "Interface", "Documentation", "Arguments", "Dependants", and "All" alternatives, respectively. Then, the average NR was 93.94%. Rounded averages are

shown above each bar of Figure 5.2. It is worth noting that the standard deviation for the 50 repetitions using the TF-ICF registry were 0.54%, 0.16%, 0.31%, and 0.47%, for NR, *R*-Precision, Recall-at-10, and Precision-at-*1*, respectively.

The achieved results shows that EasySOC performed better using TF-IDF than any other evaluated term weighting technique. The biggest gain takes place in the NR and Recall metrics. For NR metric the difference between TF-IDF results and the others was of 10 or more percent points. On the other hand, TF-ICF showed a little negative impact on Precision-at-*n* results.

These results are similar to those reported in (Kokash, 2006), where different discovery algorithms are evaluated under identical conditions showing that a TF-IDF based one outperforms the others. Accordingly, despite of TF-IDF scalability issue, we will use this combined heuristic for the experiments that are described next.

5.2.4 Comparison of query expansion alternatives

The second experiment was conducted to analyze the implications of expanding queries using terms extracted from different parts of service-oriented applications. To conduct the experiment the EasySOC registry -in particular the TF-IDF registry- was supplied with the data set. Once the data set was published, the query expansion alternatives were used to look for relevant services.

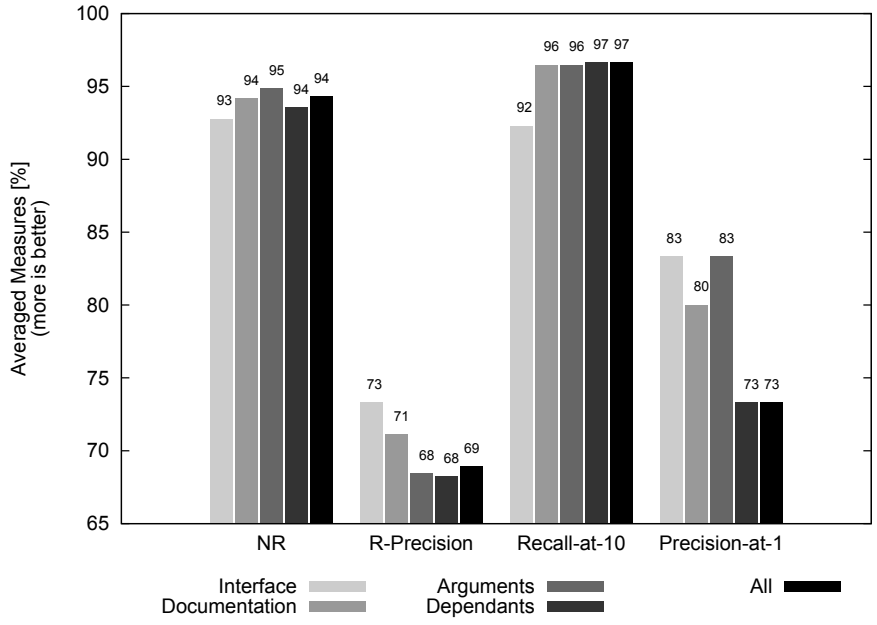

Figure 5.3: Retrieval effectiveness using different approaches to Query Expansion.

Figure 5.3 shows the achieved results. We obtained the NR, Precision-at-*n*, and Recall-at-

n measures for each query element of Table 5.2. Then, results for each query expansion alternative were calculated as the average of the results associated with its corresponding 30 queries. Then, each bar in Figure 5.3 stands for the averaged metric result that was achieved using a particular query expansion alternative.

Figure 5.3 shows that by using query expansion EasySOC retrieved more relevant documents with this data set, based upon Recall-at-*10* results. On the other side, Precision-at-*1* did not improve and *R*-Precision fell by 2.22%. These results stem from the fact that expanded queries are more general descriptions of the needs, in semantic terms. Instead, non-expanded queries have few words, resulting in very specific descriptions of the needs. Quantitatively, this is shown in Table 5.2. In the vector space context, expanded queries -more general descriptions- contain more dimensions than non-expanded queries -specific descriptions-. The inclusion of one or more dimensions sensitively impacts on cosine similarity measure results (see equation 4.4). Then, by expanding queries, services with more general textual descriptions are ranked first. Therefore, we could expect having higher Recall-at-10 -i.e., retrieving more relevant services at a cut-off of 10- than retrieving a specific relevant service at the top of the list. In addition, the fact that for 10 queries there is only 1 relevant service in the data set, severely harmed *R*-Precision when the first service retrieved was not a hit. Meanwhile, NR, which indicates a trade-off between the number of relevant services retrieved and their position in the result list, showed better performance when using query expansion.

5.2.5 Comparison of EasySOC with two related works

The third experiment was to compare the retrieval effectiveness of EasySOC with two existing approaches. For this comparison, two different approaches to service discovery were used. On one hand, Lucene (Hatcher and Gospodnetic, 2004), which is well-known open-source search software that follows a classic IR-based approach, was employed. On the other hand, an academical approach that combines classic IR techniques, term expansion based on lexical relations and structural matching techniques, was employed. For the sake of readability, in the remaining of this thesis we will refer to the latter approach as ILS (IR Lexical Structural). The details of ILS can be found in (Stroulia and Wang, 2005). Notwithstanding their differences, the three approaches return a ranked list of candidate services for a given query.

The methodology employed for this comparison was the same that was used for the second experiment described in this section. In fact, such experiment was reproduced using Lucene and ILS approaches. Concretely, the same the 391 WSDL documents were published in both Lucene and ILS registries. The evaluation set consisted of the same 150 queries as above, i.e. the 30 queries and their corresponding derivations, namely

"Interface", "Documentation", "Arguments", "Dependants" and "All". Finally, the NR, Precision-at-n, and Recall-at-n results that were achieved by Lucene and ILS alternatives were compared with those achieved using EasySOC registry, which are shown in Figure 5.3.

In order to enable a fair comparison two cautions were taken. First, the Lucene-based approach was powered with our text mining processes, so that it can deal with documents in WSDL and source code-based queries. Lucene has been designed for indexing any kind of textual documents and answering keyword phrases queries without taking into account WSDL characteristics nor common practices present at source code files. Therefore, each service description and query were preprocessed, before using them with a new discovery alternative that we called Lucene4WSDL. The reason for doing this was to evaluate whether the Lucene4WSDL alternative may be benefited from our preprocessing techniques, or not.

The other caution was in regard to the ILS registry. To the best of our knowledge there is not a publicly available implementation of the ILS approach. Thus, ILS registry was implemented basing on the authors' article (Stroulia and Wang, 2005). Broadly, ILS gathers terms from a WSDL document and represents them as three sub-vectors: (1) stems of the gathered terms, (2) stems of the synonyms of the gathered terms, and (3) stems of the words hierarchically related to the gathered terms, such as hypernyms, hyponyms, and siblings. Queries are represented using three vectors also. ILS defines the overall similarity between a service description and a query as the average of their three sub-vector matching scores. Different weights are assigned to the different sub-vector matching scores, being 3 for gathered terms, 2 for synonyms and 1 for hierarchically related. Optionally, ILS compares the structure of the candidates against a, possibly partial, WSDL specification of the desired service, which the user who performs the discovery must supply. The structural matching technique considers the names and data-types of operation messages, the number of operations per service and their corresponding names as well. As the inquiry interface of ILS optionally accepts a functional description of the desired services using WSDL, a WSDL document for representing each query in the test data set was built. To do this, a library, called Java2WSDL[1], that transparently transforms a Java interface into a WSDL document was used. Finally, ILS was tested with each query that had been derived using the query expansion alternatives plus the generated WSDL document.

Figure 5.4 depicts the averaged results. Each bar in the figure represents the result of calculating each measure with a discovery approach, and then averaging the results over the 150 queries. In this sense, Figure 5.4 summarizes the averaged results when using different term sources with each service discovery approach, like we did for the comparison

[1]Java2WSDL, http://ws.apache.org/axis/

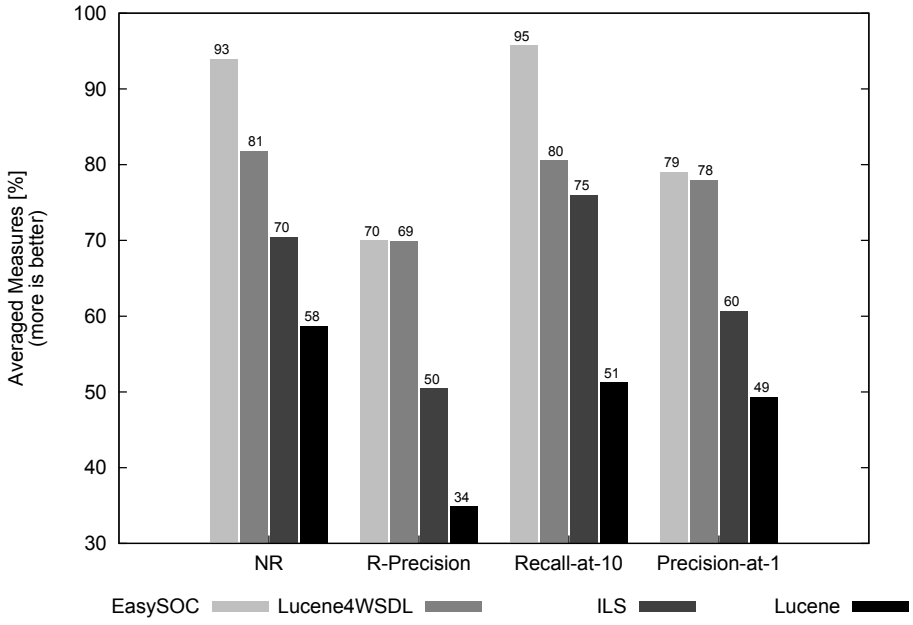

Figure 5.4: Comparison of the retrieval effectiveness of WSQBE+.

of term weighting techniques experiment.

As shown in the figure the retrieval performance of EasySOC registry surpassed that of the other evaluated registries. Moreover, the superior performance of the text mining powered Lucene variant (named Lucene4WSDL in the figure) with respect to the original one, validates the hypothesis about that helping Lucene deal with documents in WSDL and source code-based queries would improve its overall retrieval performance, making a more unbiased comparison. Accordingly, the achieved values of precision measures, such as *R*-Precision, and Precision-at-*1*, of both EasySOC and Lucene4WSDL were very close. However, EasySOC achieved Recall-at-*10* and NR of 93% and 95%, on average, whereas Lucene4WSDL achieved 81% and 80%, respectively.

The biggest gain takes place in the results for the NR measure, in which the experiments with EasySOC registry surpassed by 15%, 20% and 44% on average, its counterparts. These results confirmed that EasySOC not only retrieved more relevant services than the other evaluated approaches, but also ranked them better in the result list. In concordance with related experiments that empirically show that users tend to select higher ranked search results (Agichtein et al., 2006), even a moderate improvement in this direction has a great impact on the discovery process. For instance, the probability that a user accesses the first ranked result is 90%, whereas the probability for accessing the second ranked one is, at most, 60% (Agichtein et al., 2006). Therefore, is very important to retrieve as many WSDL documents as possible relevant to our queries, but also to rank them before non relevant ones, as EasySOC did in the reported experiments.

5.3 Discoverability anti-patterns: Survey and Experimentation

This section discusses the frequency of the discoverability anti-patterns in public Web Service descriptions, and their impact on discovery and users' ability to understand WSDL documents. The body of 391 WSDL documents that has been presented in previous sections was surveyed looking for discoverability anti-patterns. Section 5.3.1 presents survey results. Then, an experiment to provide empirical evidence of the impact of each individual anti-pattern on the retrieval effectiveness of three discovery systems was performed. We compared the retrieval effectiveness of the employed registries when feeding them with WSDL documents with anti-patterns versus improved WSDL documents, i.e. without anti-patterns. Section 5.3.2 presents the results related to the impact of the anti-pattern on discovery. Finally, another experiment consisted in asking software developers and software engineering students, who were taking a SOC[2] course at the UNICEN, about the ability of several WSDL documents to explain what the functionality offered by their corresponding services was. Answers related to WSDL documents with anti-patterns versus the answers related to improved WSDL documents were compared. Section 5.3.3 presents these results.

5.3.1 Data set analysis

The body of WSDL documents that have been gathered from Internet repositories by Heß et al. (2004), was exhaustively analyzed. For analyzing the WSDL documents of the data set a detection criterion for each WSDL anti-pattern was developed. As described in section 4.5, detecting an anti-pattern is strongly related to the way it manifests itself. Thus, some anti-patterns are harder to detect than others.

Detection criteria for Evident anti-patterns are based on the syntax of a WSDL document. A clear case of this is Enclosed data model anti-pattern, since it can be deterministically detected by applying the following rule: "if at least one type is defined in a WSDL document, then the anti-pattern occurs". Another clear case of this, is when a WSDL document has no comments.

Detection criteria for Not immediately apparent anti-patterns require to analyze not only the syntax of a document, but also its semantics comprising questions like "Is the name of this message part ambiguous?" or "Is the documentation of this operation clear enough?". The answers to this kind of questions depends on personal judgement. However, analyzing whether the names and comments present in a WSDL document follow the conventions described in section 4.5.2 may help to detect Ambiguous names and Inappropriate

[2]SOC course at the UNICEN, `http://www.exa.unicen.edu.ar/~cmateos/cos`

comments. In addition, sometimes Undercover fault information within standard messages anti-pattern has no footprint in a WSDL document. If a message includes a part to inform whether an error has occurred, this anti-pattern can be detected. Instead, if an output message part is called "parameter" and it exchanges a whatever type, it might be used to inform an error, but it is impossible to know for certain. In this case, the Undercover fault information within standard messages anti-pattern cannot be detected, unless the service implementation fires an error during a request. For the study presented in this section we considered the Undercover fault information within standard messages anti-pattern only if it can be detected in the WSDL document.

The results showed that some anti-patterns affect more WSDL documents than others, but even the least frequent anti-pattern occurs in 31 WSDL documents. Graphically,

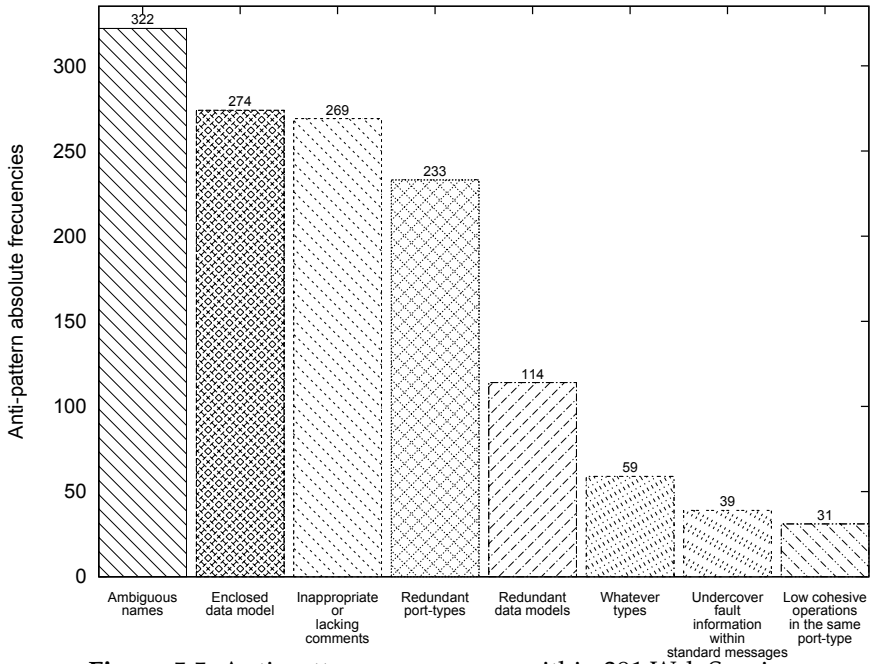

Figure 5.5: Anti-pattern occurrences within 391 Web Services.

each bar of Figure 5.5 depicts the number of WSDL documents that suffer from an anti-pattern. It is worth noting that these values represent the number of WSDL documents affected by an anti-pattern, not the number of anti-pattern occurrences. For example, if a WSDL document has 2 occurrences of the Redundant data models anti-pattern, we count as only 1 affected file. The reason to present the results in this way, is that some anti-patterns occur more than once in a WSDL document (Inappropriate or lacking comments, Ambiguous names, Redundant data models, Whatever types, Undercover fault information within standard messages), but other anti-patterns occur only once in a WSDL file (Redundant port-types, Low cohesive operations in the same port-type, Enclosed data model). Therefore, assessing the number of service descriptions that suffer from each

anti-pattern depicts how important each anti-pattern is for the surveyed data set.

Notably, though good naming and commenting practices facilitate software reuse and the fact that Web Services are developed for reuse, the fractions of documents that suffer from Ambiguous names and Inappropriate or lacking comments anti-patterns are 82% and 69%, respectively. Additionally, although enclosed data-type definitions hinder the reuse of data models, the Enclosed data model anti-pattern occurs in 70% of the documents. Similarly, notwithstanding port-types are meant to define the functionality of a service independently of any technological aspect, we have found that there are redundant protocol-dependent port-types in 60% of the documents in this data-set. On the other hand, the least common anti-pattern within this data set is the Low cohesive operations in the same port-type anti-pattern, which affects 7.6% of the documents. Likewise, the proportion of Web Service descriptions that suffers from Undercover fault information within standard messages anti-pattern is 10%.

In addition, the survey suggests that the occurrence of some anti-patterns may be related. For instance, Whatever types or Redundant data models anti-patterns never occur without Enclosed data model anti-pattern. Probably, this happens because if the developers of the service take time to separate a data model from WSDL documents, they will want to reuse that data model in other services. Therefore, in terms of understandability and discoverability, developers will strive to produce better models. However, it may be possible that the anti-pattern has not been detected because separating the data model from the WSDL helps conceal others anti-patterns. This could be the case of some types defined in two different XSD files, i.e., a redundant data model, and then imported from a WSDL document.

Another case where anti-patterns are related is the Uncovered fault information within standard messages anti-pattern with the Ambiguous names anti-pattern. As shown in the example in section 4.5.8, to inform that an error has occurred in an output message may require control coupling (Yourdon and Constantine, 1979). The names of the parameters, when this kind of coupling is present, usually take the form of <verb>+<noun>, e.g., “isError”. And that is an instance of the Ambiguous names anti-pattern.

5.3.2 Measurement of anti-patterns impact on discovery

In order to provide empirical evidence of the impact of each individual anti-pattern on the retrieval effectiveness of three discovery systems, an experiment was performed. The experiment consisted in comparing the effectiveness of three approaches to service discovery when using different versions of a data set of WSDL documents. The employed versions of the body of WSDL documents were built by removing anti-patterns. One version consisted of the original WSDL documents. Another version consisted of the

WSDL documents without anti-patterns, i.e. after removing all found anti-pattern occurrences. Moreover, one version of the data-set per anti-pattern was built, in which the corresponding anti-pattern was removed from all the WSDL documents.

With respect to the data set, the same data set as before was used to build 9 versions of the 391 WSDL documents (the original version plus eight new versions). Seven versions of the data-set were built by removing each anti-pattern individually. To do this, an experienced service-oriented application developer identified which anti-patterns were present in a given WSDL document. Then, he looked for their solutions in section 4.5, and built as many versions of the given WSDL document as anti-patterns it had by separately removing the occurrences of each detected problem. Another version of the data set was built by removing all the anti-patterns of each WSDL document.

It is worth noting that removing the Enclosed data model anti-pattern, and building the corresponding data set version, was omitted. The reason behind this is that the same retrieval results are achieved by either enclosing data-set models or moving them to a separated schema file. This particular anti-pattern actually impacts on discovery when data models are designed to be reused and best practices for modelling data are followed, which requires to place these models on separated files. Nevertheless, this experiment measured to some extent the impact of these best practices since they are within the scope of other anti-patterns, like Ambiguous names or Redundant data models.

The methodology followed to perform the experiment comprised publishing each version of the data set in the service registries, performing queries and analyzing whether the retrieved services were relevant to the queries or not. The employed registries were Lucene4WSDL, ILS, and EasySOC. The evaluation set consisted of the "Documentation" queries described earlier in this chapter.

The NR, Precision-at-n, and Recall-at-n metrics were calculated for each query and each registry using the 9 versions of the data set, and then averaged the results over the 30 "Documentation" queries per registry. The obtained results showed that separately remedying 3 of the identified anti-patterns, namely Whatever types, Undercover fault information within standard messages and Low cohesive operations of the same port-type, had an insignificant impact on the retrieval effectiveness of the employed registries. These results may stem from the fact that these anti-patterns are the least frequent among the analyzed WSDL documents. Specifically, the Whatever types anti-pattern affects 63 WSDL documents, the Undercover fault information within standard messages anti-pattern occurs in 59 documents, and the Low cohesive operations of the same port-type anti-pattern in 31 documents (see Figure 5.5).

Figures 5.6, 5.7 and 5.8 show the average results of each metric when using the data sets with Lucene4WSDL, EasySOC and ILS registries, respectively. In order to enable comparisons, the results are arranged in groups of six bars within each figure, in which each

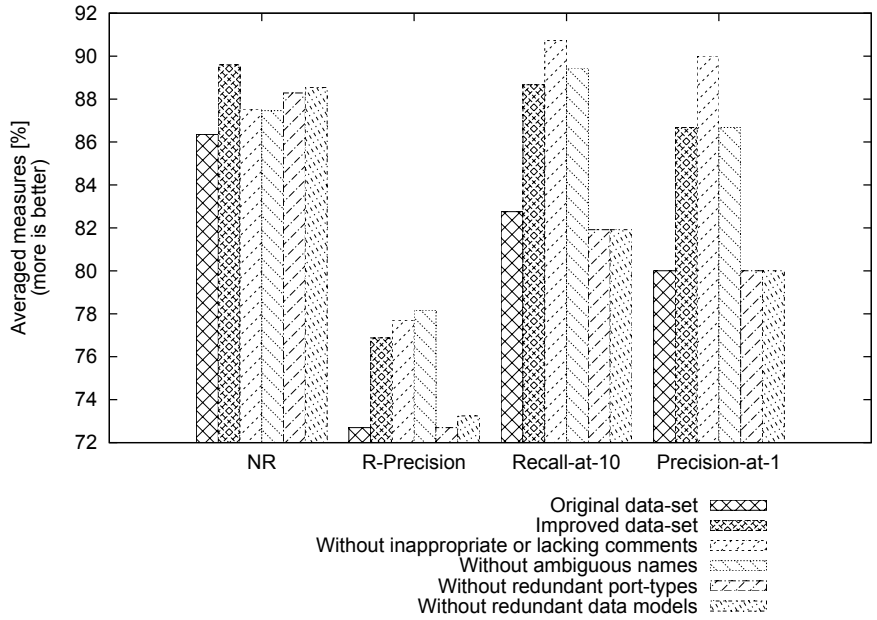

Figure 5.6: Average measure results using Lucene4WSDL registry.

group is associated with an employed metric. From left to right, the first bar within each group represents the achieved results for each metric when using the original version of the data set, the second bar represents the results when removing all anti-patterns from the data set, the third bar represents the results when removing the Inappropriate or lacking comments anti-pattern, the fourth bar represents the results when removing the Ambiguous names anti-pattern, the fifth bar represents the results when removing the Redundant port-types anti-pattern, and the sixth bar represents the results when removing the Redundant data models anti-pattern.

In general, Figures 5.6, 5.7 and 5.8 show that the evaluated registries achieved better performance measures when using any of the improved versions of the data set than when using the original version. The biggest gain takes place in the results associated with the Precision-at-*1* measure. The experiments when removing the occurrences of the Inappropriate or lacking comments anti-pattern showed that the Lucene4WSDL, EasySOC and ILS registries improved their Precision-at-*1* by 10%, 13.33%, 3.33%, respectively. Figures 5.6, 5.7 and 5.8 show that the tendency to improve Precision-at-*1* results was maintained by the removal of the other anti-patterns as well. Having a higher Precision-at-*1* means that the registry performs better in retrieving a relevant service at the top of the result list.

The *R*-Precision results have provided more evidence of the improvements in the retrieval of relevant services before non-relevant ones when removing anti-patterns from the data-set. In addition, the Recall-at-*10* results have empirically shown that the em-

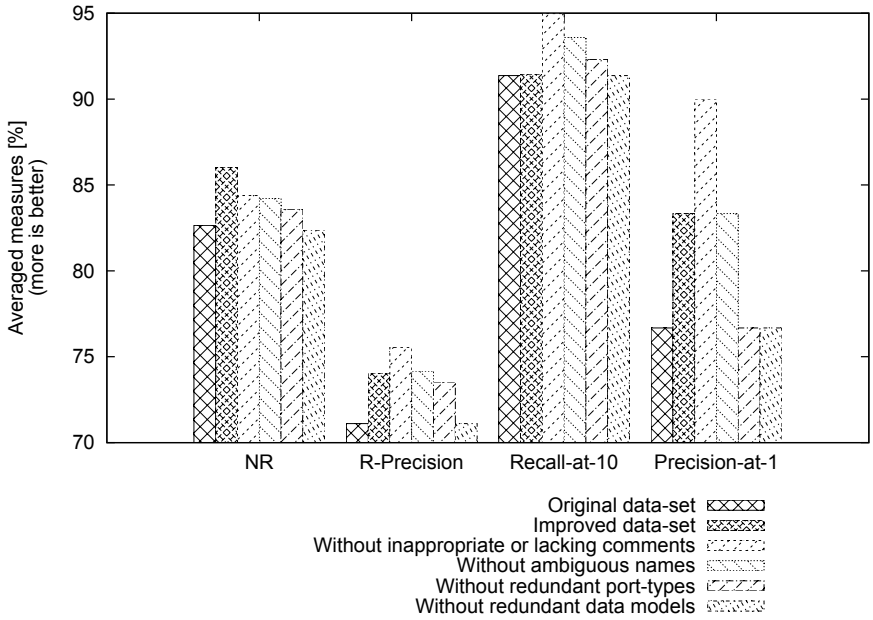

Figure 5.7: Average measure results using EasySOC registry.

ployed registries were more effective in retrieving more relevant services when using the improved data-sets. Moreover, when removing all the anti-patterns the employed registries have empirically shown to improve Normalized Recall. As this measure takes into account Recall and the position of each relevant retrieved service within the result list, the results confirmed that the employed registries not only retrieved more relevant services, but also ranked them first in the result list when using the WSDL documents without any anti-pattern.

These positive results may stem from the fact that the original WSDL documents usually contain redundant, meaningless and nonspecific terms, i.e., when there is no a strong connection between the operation functionality and the terms used to define it (e.g., "calculate(int i0, int i1):int"). For example, the term "parameters" is frequently used for naming inputs and "return" for outputs, whereas "body" is usually associated with operations bound to HTTP protocol. Table 5.3 lists the part names most frequently encountered within the original version of the studied data set.

Specifically, the original WSDL documents have 3368 unique terms, but after removing the identified anti-patterns they only have 2555 unique terms. Indeed, as the proposed anti-pattern solutions remove meaningless nonspecific terms and add representative names, the refactored WSDL documents have less terms but they are more representative of the functionality of the services. This kind of terms strongly impacts the retrieval performance of syntactic approaches to service discovery (Dong et al., 2004; Kokash et al., 2006; Jian and Zhaohui, 2005). Therefore, we could expect retrieving a specific relevant

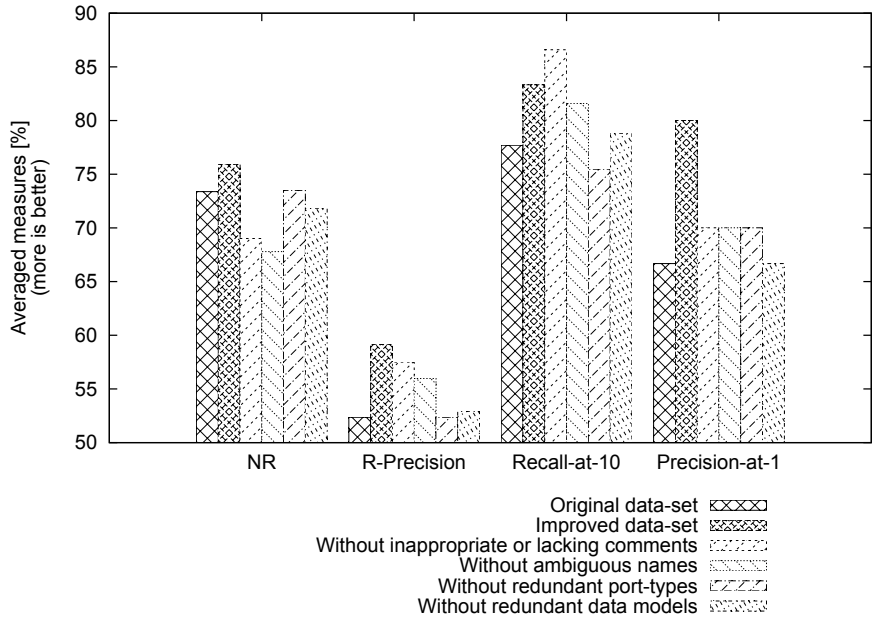

Figure 5.8: Average measure results using ILS registry.

Name	Occurrences	Frequency
parameters	2124	16.32
body	2044	15.57
return	524	3.99
password	473	3.6
userid	275	2.09

Table 5.3: Commonest message part names.

service near to the top of the list when specific and representative service descriptions have been published in the registry, which is the case when using the improved WSDL documents.

5.3.3 Measurement of anti-patterns impact on users' understanding

The second experiment consisted in asking software developers and software engineering students, who were taking a SOC[3] course at the UNICEN, about the ability of several WSDL documents to explain what the functionality offered by their corresponding services was. Answers related to WSDL documents with anti-patterns versus the answers related to improved WSDL documents were compared. The results shown a major ten-

[3]SOC course at the UNICEN, `http://www.exa.unicen.edu.ar/~cmateos/cos`

dency in the answers, specifically 84.62% affirmed that the improved WSDL documents were more intelligible than the original ones.

Concretely, the participants were given several WSDL documents and a questionnaire designed to analyze the implications of those anti-patterns that occur less frequently among the WSDL documents of the analyzed data set, namely Low cohesive operations of the same port-type, Undercover fault information within standard messages, Whatever types and Redundant data models.

To build the employed WSDL documents, service descriptions that have the aforementioned 4 anti-patterns were taken from the data set used for the discovery experiment. Then, for each service an improved WSDL document was generated by removing the anti-pattern occurrences. The participants answered several questions about the readability of original version of each WSDL document. Afterward, they were asked about the improved versions. The questionnaire was given as a homework to 26 participants who were taking a SOC course, and the data were collected on the 18^{th} of September. The group of participants was integrated by last year software engineering students and practicing software engineers. It is worth noting that the participants did not know about the catalog of discoverability anti-patterns proposed in this paper or its related works, until the survey was finished.

The questionnaire consisted of eleven questions divided in three groups. A group of questions were designed to familiarize the participants with each version of the employed WSDL documents. For example, a question asked about the number of operations offered by a service. Another group of questions asked the participants about whether the WSDL documents were explanatory enough to they understand what the offered service does and how to use it, or if their descriptiveness could be improved to some extent. The last group of questions allowed participants to comment which version of the employed WSDL documents would outsource and why. The questions of the second and third groups, and the main results of the survey are described next.

First, the participants were given a service port-type with several operations belonging to the same domain, but one operation of a different domain, and in turn asked the participants whether removing the non-cohesive operation would improve the understandability of the service or not. In other words, we were implicitly asking individuals if they conceived Low cohesive operations of the same port-type anti-pattern as a problem that would hinder understanding the service, and if they would apply its associated remedy. The results showed that 92% of the participants implicitly answered that they would remove the anti-pattern.

Second, the participants were given a service operation that returns a data-type based on the xsd:any constructor, and whose documentation provides hints that, in case of an execution problem, error information would be included in the output message. Then,

three questions were asked to the participants. The first question was about whether they could determine the structure of the operation response. The second one asked them about whether they would replace the data-type of the operation output with a data-type that merely represents the operation result. Finally, the third question was if they preferred piggybacking error information in output messages or exchanging it in fault messages. In other words, we implicitly asked the participants to evaluate whether the Whatever types and Undercover fault information within standard messages anti-patterns took place in the analyzed WSDL document, and whether they would apply the proposed remedies or not.

As a result, 92% of the participants answered that the structure of the output could not be determined. The rest of the participants answered that the analyzed operation always returns instances of xsd:double or xsd:int data-types. This result may stem from the fact that the operation is for uploading files, and if a file is successfully transferred, then the file size is returned. In this sense, it seems that 8% of the participants disregarded the possibility of a failure during the execution of the operation. The results also showed that 92% of the participants would replace the data-type of the output of the operation. As the reader can see, the percentage of participants that identified the Whatever type problem was exactly the same that voted for replacing the data-type definition.

On the other hand, 92% of the participants realized that the analyzed output message could exchange error information. However, 81% of the them answered that they would use fault message to convey error information. This may be related to the fact that fault messages are curiously uncommon among the WSDL documents of the data set.

In order to collect opinions about the Redundant data models anti-pattern, the participants were given a WSDL document with two operations returning the same data-types, but defined twice. The participants were asked whether they would remove one of the redundant data-types or not. Eighty one percent of them answered that they would remove the repeated data-types.

For the next group of questions, the participants were given the improved version of each WSDL document that they had analyzed. Finally, we asked the participants about which version of the analyzed WSDL documents they would outsource. Additionally, each participant gave his/her opinions about why they would select one or another version. The results showed that 84% of the participants would use the improved WSDL document, 8% the original version and 8% any version indistinctly.

The comments made by the participants provided an idea of the reasons behind choosing the improved versions. Some participants included two, or more, different reasons in their comments. From these comments we summarized and ranked the most frequent main reasons. Accordingly, the reasons are listed in decreasing order of occurrence next:

1. the data-types exchanged by the improved WSDL document were better represented,
2. the improved WSDL document was more concise,
3. the operations of the improved WSDL document belonged to a single domain,
4. error information was better handled by the improved WSDL document.

Specifically, the results showed that 16 participants included in their comments the reason related to exchanged data-type definitions. The responses of 13 participants highlighted that the improved WSDL documents were more concise than the original versions. Eight participants commented that they would outsource the improved WSDL documents because they arranged cohesive operations. Finally, 6 participants said that they would outsource the improved versions because separating error information from output messages helped them to understand how to access the service.

As the reader can see, the reasons given by the participants express the results of remedying some of the identified anti-patterns. In particular, improving the XSDs of the data-types exchanged by the analyzed services, which was ranked first, was caused by remedying the Redundant data models and Enclosed data model anti-patterns. These anti-patterns are also associated with the second reason of the rank, which takes under consideration the length of the analyzed WSDL documents, given that remedying both aforementioned anti-patterns causes conciser WSDL documents. The other two reasons given by the participants are related to remedying the Low cohesive operations in the same port-type and Undercover fault information within standard messages anti-patterns, respectively.

5.4 Memory usage and response time of EasySOC implementation

Finally, some experiments were conducted to measured the performance of current EasySOC registry implementation. First, the overheads introduced by EasySOC into publication and discovery tasks were measured. The average overhead introduced by EasySOC when publishing a new Web Service has been empirically shown to be 14.34 milliseconds (ms) over doing this task without EasySOC. Moreover, the overhead introduced by EasySOC when discovering Web services has been empirically shown to be 13 ms, on the average case.

Both publishing and discovering evaluations were deployed on an Intel Pentium D working at 3.0 GHz with 1.0 GB of RAM. Here, EasySOC and jUDDI [4] registries were running

[4] jUDDI `http://ws.apache.org/juddi/`

on Sun JVM 1.6.0_02 and Ubuntu Linux 7.10. It is worth noting that both EasySOC and jUDDI along with the data set were hosted at the same computer, to avoid the overhead introduced by the network. Clearly, in a real word scenario, the Web Service registry, publishers and discoverers may be spread over the Internet. Conversely, EasySOC and an UDDI registry often reside at the same server machine or, at least, at the same local area network, in which the cost of networking is inexpensive. Then, the cost associated with forwarding an UDDI request from EasySOC can be safely ignored.

On one hand, to measure the time penalty for publishing a WSDL document, the time required for gathering relevant information, i.e. processing, from each document of the data set was measured. The processing time required by UDDI was not measured, because we wanted to analyze how EasySOC registry impacted over the performance of classic publication and discovery tasks. In order to mitigate any noise introduced by external conditions that could influence this performance test, each WSDL file was processed 10 times. The CPU time demanded by each preprocessing execution was measured and the average was computed. As mentioned, the average was 14.34 ms.

On the other hand, to evaluate the overhead introduced when discovering services we measured the time required by EasySOC to answer the 30 "Documentation" queries, which were described in previous sections. Again, 10 executions were performed for each query, and then their average was computed. As mentioned the average was 13 ms.

Moreover, a second experiment consisted of evaluating the scalability of EasySOC. This experiment concerned memory usage and response time when the number of published Web Services grows. For this experiment the same deployment settings as before were used.

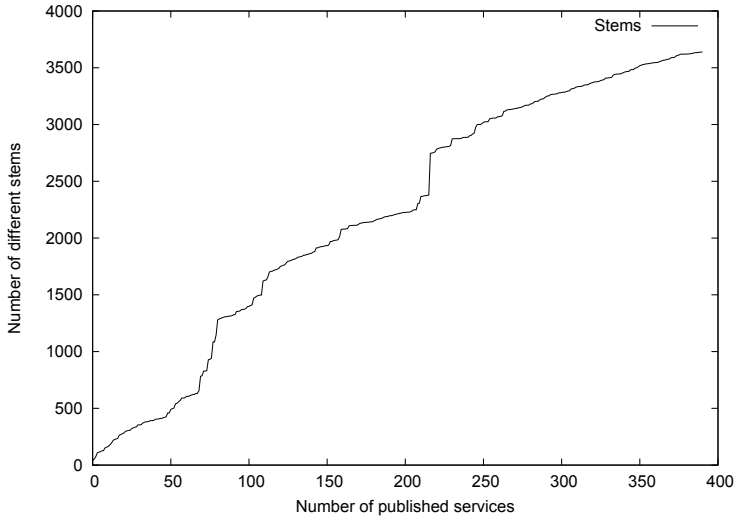

Figure 5.9: Number of stems when the number of services grows.

The averaged memory usage of the JVM has been empirically shown to be 9.76 Megabytes (Mb), with a standard deviation of 2.78 Mb. This results may stem from the following facts. The building block of EasySOC implementation is a "vector", in which each entry is a weighted term. Then, a Vector Space is a collection of vectors and a partitioned Vector Space is a collection of such Vector Spaces. In consequence, it is expected that the more terms, the more memory usage.

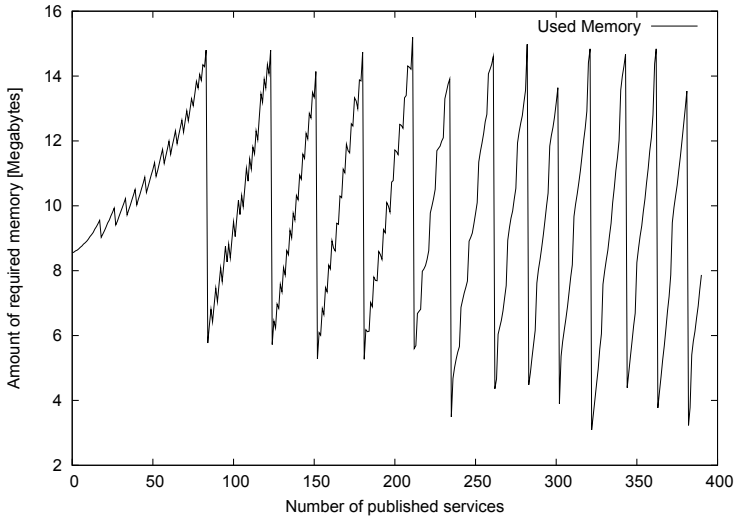

Figure 5.10: Memory usage when the number of services grows.

Figure 5.9 shows the correlation between the number of published Web Services (x axis) and the number of different stems (y axis), while Figure 5.10 shows the memory usage variations. As shown in Figure 5.9, initially the number of different stems linearly grows with the number of published services. However, the number of stems converges on the sum of different terms of each individual category or sub-language. With respect to memory usage, the experiments have shown that EasySOC registry had a consumption peak of 15.2 Mb with 211 published services. Afterward, the memory usage oscillated between 4 Mb and 15 Mb. As shown in Figure 5.10 memory usage was influenced by the underlining JVM mechanism for recycling unused objects (a.k.a. garbage collector).

On the other hand, to measure the impact of the number of published services on response time, n services were arbitrarily picked and published in EasySOC registry. Then, the time required by EasySOC to answer the aforementioned 30 queries was measured. Response time was measured for each query with seven values of n: $n = 60$, $n = 120$, $n = 180$, $n = 240$, $n = 300$, $n = 360$ and $n = 391$. Again, 10 data-sets for each value of n were built.

Figure 5.11 shows the resulting times for each value of n per query. As the reader can see, the more published services, the more vector comparisons, and therefore the more response time for a discovery request. As expected, the JVM memory management im-

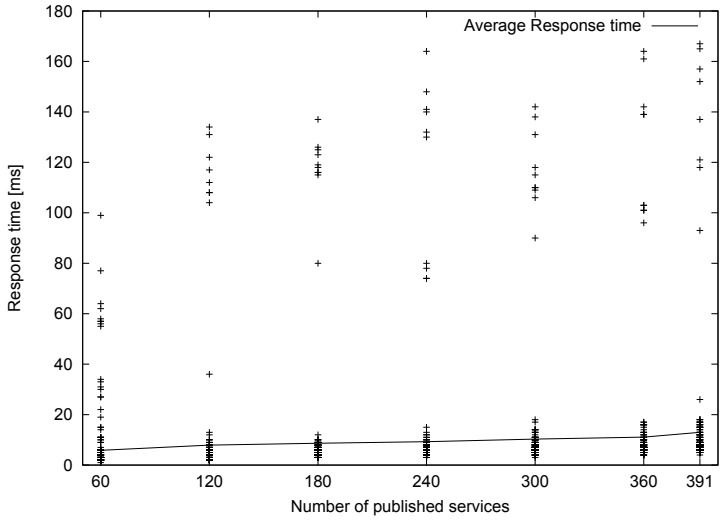

Figure 5.11: Response time when the number of services grows (full scale).

pacted on the variability of these results as well. For the sake of readability, Figure 5.12 shows the averaged results.

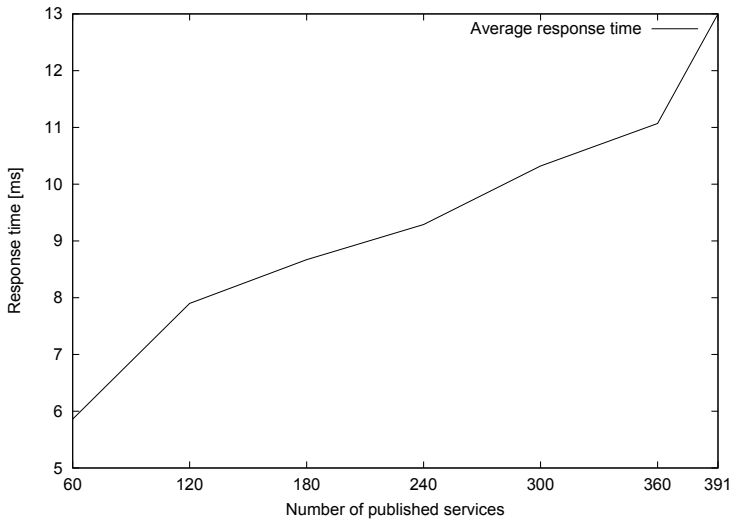

Figure 5.12: Response time when the number of services grows (only average).

Finally, the time required to build and answer queries of different size was measured. For this experiments the 150 queries that were previously described, were employed. In order to mitigate any noise introduced by external conditions that could influence this performance test, 100 executions for each query were performed and then the average was computed. The averaged overheads introduced by EasySOC when discovering Web Services were 8.56 ms, 164.45 ms, 362.33 ms, 281.48 ms and 468.59 ms using Interface, Documentation, Arguments, Dependants and All query expansion alternatives, respec-

tively. This evaluation was executed on the same Intel Pentium D working at 3.0 GHz with 1.0 GB of RAM, running Sun JVM 1.6.0_02 and Ubuntu Linux 7.10.

5.5 Conclusions

This chapter described in detail some experiments that were performed with current implementations of the EasySOC registry and plug-in, in order to validate the underlying approach. Specifically, the validation comprised four groups of experiments.

First, EasySOC search space reduction support was compared with K-NN, Naïve Bayes, and a combination of Naïve Bayes and Support Vector Machine algorithms. Experiments show that the EasySOC search space reduction support was more accurate than its counterparts, at least under the employed settings.

Second, the retrieval effectiveness of the EasySOC registry was evaluated by means of three experiments. The first experiment within this group was intended to assess EasySOC retrieval effectiveness when varying the employed term weighting technique. Achieved results corroborated that TF-IDF surpassed other techniques. The second experiment aimed to assess EasySOC retrieval effectiveness with different query expansion alternatives. Achieved results pointed that by following the Query-By-Example metaphor specific queries with less terms than expanded queries were generated. Thus, using non expanded queries specific results are ranked first, wheres using more general queries the possibilities of including a relevant service at the top of the list decrease as the possibilities of including such relevant service before the 11^{th} positions augment. The last experiment within this group was a comparison of EasySOC with two related work, in terms of retrieval effectiveness. Achieved results have empirically shown that EasySOC not only included more relevant services in the candidate list, but also ranked them first.

Third, another experiment group was related to discoverability anti-patterns, and measuring their implications on discovery. The retrieval effectiveness of three discovery mechanisms was assessed when using the original WSDL documents and the improved ones, i.e., the WSDL documents that have been refactored according to the proposed solution of each anti-pattern. The fact that the results related to the improved data sets surpassed those achieved by using the original data set regardless the approaches to service discovery employed, provided empirical evidence that suggests that the improvements are explained by the removal of discoverability anti-patterns rather than the incidence of the underlying discovery mechanism. A survey was also performed, in order to collect subjective opinions from software engineering students and practicing engineers about how intelligible several WSDL documents were. The results of the survey suggested that WSDL documents should be improved to increase service chances of being understood and, in turn, outsourced.

Fourth and finally, the memory usage and response time of current EasySOC implementation were measured. Although the implementation of EasySOC is not optimized, for example, it neither uses indexes nor takes advantage of multi-core processors for comparing vectors in parallel, achieved results have empirically corroborated its feasibility.

Chapter 6

Conclusions and Future Work

Service-oriented software systems have recently emerged from component-based software engineering supporting a new paradigm, which is more aligned with business concepts and allows for dynamic composition of software architectures. Service-oriented software systems started as distributed or web-based applications, but are now spreading also into a new wave of service-oriented software systems, such as Cloud computing (Buyya et al., 2009) and Software-As-A-Service (Turner et al., 2003) systems.

The success encountered by the Web has shown that loosely coupled software systems can be more flexible, more adaptive and often more appropriate in practice than tightly coupled software systems. Loose coupling makes it easier for a given system to interact with other systems, possibly legacy systems that share very little with it. Web Services are at the crossing of distributed computing and loosely coupled systems.

When properly implemented, Web Services can be discovered and reused dynamically using non-proprietary mechanisms, while each service can still be implemented in a black-box manner. This is important from a business perspective since each service can be implemented using any technology, independently of the others. What matters is that everybody agrees on the integration technology, and there is a consensus about this in today's middleware market: customers want to use Web technologies (Martin-Flatin and Löwe, 2007).

Distributed applications may gain high levels of interoperability, flexibility and reuse by relying on Web Service technologies. However, as the amount of published Web Services grows, discovering proper services becomes harder. While concepts for service design (Erl, 2007), composition, versioning (Juric et al., 2009), and management are currently maturing, an approach to efficient and easy to adopt discovery is still missing in the literature (Garofalakis et al., 2006). Therefore, this thesis is focused on this problem.

6.1 Contributions

This thesis proposed the EasySOC approach. Central to the proposed approach is the fact that properly described Web Services and consumers' applications, i.e., client-side applications designed to consume third-party services, may convey information for bridging the gap between services and discoverers. Specifically, the WSDL document and UDDI entry of a service may convey important information about its offered functionality, such as the name and comment of each offered operation, or the category of the service, which may help to enhance its discoverability. At the same time, the source code associated with client-side applications may carry information about the functional descriptions of the potential services that can be discovered and, in turn, consumed from within those applications.

Therefore, this thesis introduces the following core contributions into the context of service-oriented applications development:

- A novel text mining process for gathering relevant information from existing WSDL documents. The text mining process has been designed to deal with the characteristics of the WSDL, e.g. its flexibility to represent semantically equal concepts in syntactically different forms, which degrade the retrieval effectiveness of syntactic-based registries. Accordingly, the text mining process allows gathering relevant information from WSDL documents, while bridging different naming tendencies, coding conventions, and message encapsulation styles.

- A novel text mining process for gathering relevant information and building queries from existing consumers' applications source code. The process bases on Query-by-Example and Query Expansion ideas. First, since developers commonly use abstract interfaces to represent the external services offered functionality, the process gathers relevant information from such interfaces because they exemplify the functionality required by the developers. Second, the process expands the initial query by gathering relevant information from components that directly interact to such abstract interfaces, because such components are intended to interact to the external services at run-time, actually. Accordingly, the process frees developers of generating queries.

- A novel algorithm for discovering services. The algorithm employs a Web Services classification system for automatically reducing the search space, which makes Web Service retrieval computationally efficient even with many available services. This classification system has also shown to be useful for publishers, by automatically suggesting how to fill UDDI Yellow pages. This algorithm along with the text mining process for WSDL documents have been materialized in the EasySOC registry.

In terms of the criteria described in section 3.5, the registry supports three alternatives for building queries, namely *keyword*, *natural language*, *template*, supports *syntactic mediation*, shows respectable scalability values (*number of categories*) and follows WSDL and UDDI standards as well.

- A novel catalog of Web Service discoverability anti-patterns. The catalog supplies developers with a succinct name to convey the essence of each anti-pattern, a classification of each anti-pattern related to which aspect of WSDL documents it affects, another classification based on how an anti-pattern manifests itself, examples, and a sound and practical solution for each anti-pattern. Besides, this thesis provides a guideline to employ the catalog improving WSDL documents readability and discoverability.

This thesis collaterally introduces two additional contributions into the context of service-oriented applications development:

- A reusable set of criteria for the characterization of service discovery systems, along with a survey of Web Services and Semantic Web Services discovery system alternatives, and a detailed analysis of the surveyed alternatives.
- A novel tool for developing service-oriented applications. The tool implements the query generation process described in this thesis. Once a query is built, the tool connects to a service registry and presents the list of candidates to the developer. This tool has been integrated to the Eclipse IDE, one of the most popular development environments.

6.2 Limitations

One limitation of the EasySOC approach to discover services is that its heuristic for comparing two Web Services is still syntactic though it attempts to bridge syntactic differences. For instance, suppose two Web Service operations for IP location. One takes "ip address" as input and returns "latitude / longitude". The other takes "hostname" as input and returns "zipcode". Clearly, EasySOC will place these operations far apart in the vector space model, deteriorating discovering effectiveness, unless the names of these operations and their documentation near the vectors.

Another limitation of the EasySOC approach to discover services is that it does not support service composition yet. In other words, the approach does not compose already existing services into one or more new services, when a discoverer's request can not be directly satisfied by a unique service.

A limitation of the EasySOC approach to build queries arises as a consequence of the assumptions made when developing service-oriented applications. Specifically, as suggested by (Spinellis, 2008) and (Witte et al., 2008), EasySOC assumes that developers follow best practices to name and comment their source codes. However, when such source code files are cryptic or have not comments the EasySOC approach to build queries lacks its vital input.

One weak point of the implementation of EasySOC approach to reduce the search space is that it assumes that a corpus of previously classified services is available. This generates the inability for handling dynamic creation of categories without re-building the classifier. To cope with such a requirement, an incremental clustering (Baeza-Yates and Ribeiro-Neto, 1999) approach might be more suitable than a classification one.

The EasySOC discovery algorithm has been designed to be concurrently executed using as many processors as sub-spaces exist. However, another weak point of the implementation of EasySOC registry is that its algorithm operates in a centralized way. In the near future, the EasySOC registry will be integrated to a P2P middleware to decentralize its algorithms. This point deserves to be carefully investigated because it requires to employ a decentralized term weighting scheme as well, instead of using the TF-IDF scheme, which as reported in section 5.2 may degrade the discovery retrieval effectiveness.

6.3 Future Work

This work represents a step towards answering an important research question: can realistic abstract functional specifications of service-oriented applications be converted into ready-to-run concrete applications with few human intervention? Here, the term realistic is used to differentiate such abstract functional specifications from those requiring the specification of every aspect of potential services and/or clients' applications, which may be very hard to accomplish and thus they are not present in practice.

In order to answer the aforementioned question more effort should be placed. Concretely, there are four main research lines that will be explored in the near future, which are described below.

6.3.1 Experimental evaluations

First, more experiments should be done in order to provide more evidence of the effectiveness of EasySOC approach to service discovery. Although EasySOC has been evaluated with a well known set of Web Services, the approach should be tested with other data sets, several software development teams, and larger service-oriented applications.

In this direction, an experiment related to discovery will be conducted. The experiment methodology will be the same as described through section 5.2.5, but a recently published repository of ca. 5000 real Web Services[1] will be employed. Preliminary results are encouraging and suggest that using the EasySOC plug-in is also beneficial from a practical perspective, as it allows discoverers to quickly find proper services.

6.3.2 Service consumption

Second, the EasySOC approach should be extended to incorporate another activity of the SOC paradigm for developing service-oriented applications, namely "Bind". SOC promotes the "Publish, Discover, Bind" approach to develop applications. This thesis focused on easing Publish and Discover parts of the approach. The Bind part relates to the consumption of selected services.

As shown in (Crasso et al., 2010a), current approaches to service consumption from within applications require developers to manually look for suitable services and "glue" them in their client-side code afterward. The approach commonly used by developers to "glue" a Web Service, consists in obtaining the WSDL document of the service, interpreting it, and generating a client-side proxy to the remote service. Though this approach allows designers to separate business logic from the code for invoking services, the application logic mixes up with code that is subordinated to particular service interfaces. This fact reduces the internal quality of the resulting software, in which modifiability and out-of-the-box testing (i.e. outside a SOC setting) are compromised.

Then, a line of future research will aim to let developers to focus on implementing and testing the functional code of an application, and then "SOC-enable" it by discovering and loosely assembling the external functionality. A key part for this vision, is the provision of assistance to developers for adapting specific service contracts to abstract ones. Concretely, the idea is to automate the task of bridging the signatures of the methods implemented by a service adapter and the operations of its associated real service. This technique will identify structural (e.g. parameter types/ordering, missing/extra parameters) and protocol differences between two interfaces and automatically generate bridging code accordingly.

6.3.3 Service replaceability

Third, once the main activities of the SOC paradigm for developing service-oriented applications will be studied and their problems faced, the focus should be put on problems

[1] QWS data set, http://www.uoguelph.ca/~qmahmoud/qws/

related to service replaceability. As suggested by Grefen et. al, developers of service-oriented applications might live in a highly dynamic environment, in which "Competition has become fiercer because of developments like the globalization of business, the advent of e-commerce and increased market transparency. Products and services to be delivered to customers are on the one hand of quickly increasing complexity, and on the other hand subject to increasingly frequent modifications and replacements." (Grefen, 2006). In such an environment, the replacement of services will be a very common task.

As part of the ongoing ANPCyT project (grant PAE-PICT-2007-002312), the approach described in this thesis will be integrated to the approach for component replaceability published in (Flores and Polo, 2009). The goal is to investigate whether a test-based approach to component replaceability can be used for Web Service replaceability. Flores et. al's approach takes as input a test suite specification associated with a component, a service in this case, being replaced. Assuming that good practices for software development are followed, test suite specifications can also be considered as part of realistic abstract functional specifications of service-oriented applications.

Then, a short-term research goal is to evaluate whether a test suite specification and the approach described in (Flores and Polo, 2009) can be employed to refine the list of candidates returned by the EasySOC plug-in. The idea is to automatically remove from the list of candidates those services that do not pass the given test cases. A long-term research goal is the development of an approach to discover services that not only considers service functional descriptions but also their behaviour.

6.3.4 Service descriptions with lightweight semantic annotations

By assuming that services are precisely described, i.e., augmented with semantic annotations, it is expected that automatically gaining access to Web Services would be simplified, notwithstanding the ever-increasing large and heterogeneous service space (Mateos et al., 2006). However, the important effort required from publishers to add semantics in Web Services hinders the widespread usage of Semantic Web Services is (Crasso et al., 2008b). To cope with the task of incorporating services into the Semantic Web, several approaches have been proposed for semi-automatically or automatically annotating Web Service operations and their arguments with "lightweight semantic annotations" (Kiyavitskaya et al., 2009), such as tags. Tags lightweightness with respect to ontology definition languages, encourages developers to annotate their services.

In this line of research, a novel approach called Automatic Web Service Annotator (AWSA) is described in (Crasso et al., 2010b). The approach is intended for generating preliminary annotations, meant to be revised by a human developer, of the operations associated with a Web Service. Broadly, given a non semantic Web Service, the idea is to use certain in-

formation that is implicitly conveyed in its description to identify concepts from shared ontologies that have been used for annotating services similar to the former, and then guide the mapping of its argument definitions onto suitable concepts.

6.4 Final Remarks

The Service-Oriented Computing paradigm is emerging as a new way to engineer applications that are exposed as services for possible use through standardized protocols. Service-oriented applications are pushing traditional software engineering problems, such as distribution, composition, and evolution, to their extreme. Services are developed, deployed, and evolved by different organizations. Services are exposed for possible use by other parties who may search and discover them.

To make SOC a reality, many approaches have been developed to bridge the gap between service providers and service consumers. Mostly of them assume that service descriptions are well written, or that developers enrich them with explicit semantics, which is not necessarily true (McCool, 2005). On the other hand, this thesis presents a study of how a set of real world service descriptions look like, pointing out recurrent discoverability problems but also suggesting practical solutions to them. This thesis proposed that revised service descriptions and consumers' applications designed to consume third-party services, may convey information relevant to the discovery process. This thesis presents a novel approach to discover services that exploits such relevant information, enabling service consumers to efficiently discover proper services.

6.5 Publications

The results obtained during the development of this research have been presented and published in different forums. Below, the most relevant publications are listed according to their publication date. For those articles that are still in press, the acceptance notification date was used.

6.5.1 Journals

- "AWSC: An approach to Web service classification based on machine learning techniques". Marco Crasso, Alejandro Zunino, Marcelo Campo. *INTELIGENCIA ARTIFICIAL*, ISSN 1137-3601, vol. 12, tomo 37, pp. 25-36, Asociación Española para la Inteligencia Artificial, Valencia, España, 2008.

  - Abstract: A Web service is a Web accessible software that can be published, located and invoked by using standard Web protocols. Automatically determining the category of a Web service, from several pre-defined categories, is an important problem with many applications such as service discovery, semantic annotation and service matching. This paper describes AWSC (Automatic Web Service Classification), an automatic classifier of Web service descriptions. AWSC exploits the connections between the category of a Web service and the information commonly found in standard descriptions. In addition, AWSC bridges different styles for describing services by combining text mining and machine learning techniques. Experimental evaluations show that this combination helps our classification system at improving its precision. In addition, we report an experimental comparison of AWSC with a related work.

- "Easy Web Service Discovery: A Query-by-Example Approach". Marco Crasso, Alejandro Zunino, Marcelo Campo. *Science of computer programming*, ISSN 0167-6423, vol. 71, tomo 2, pp. 144-164, Elsevier Science, Amsterdam, The Netherlands, 2008. **(Indexed by SCI)**

  - Abstract: Web services have acquired enormous popularity among software developers. This popularity has motivated developers to publish a large number of Web service descriptions in UDDI registries. Although these registries provide search facilities, they are still rather difficult to use and often require service consumers to spend too much time manually browsing and selecting service descriptions. This paper presents a novel search method for Web services called WSQBE that aims at both easing query specification and assisting discoverers by returning a short and accurate list of candidate services. In contrast with previous approaches, WSQBE discovery process is based on an automatic search space reduction mechanism that makes this approach more efficient. Empirical evaluations of WSQBE search space reduction mechanism, retrieval performance, processing time and memory usage, using a registry with 391 service descriptions, are presented.

- "Combining Query-by-Example and Query Expansion for Simplifying Web Service Discovery". Marco Crasso, Alejandro Zunino, Marcelo Campo. *Information Systems Frontiers*, ISSN 1387-3326, in press, Springer, Amsterdam, The Netherlands, Accepted 2009. **(Indexed by SCI)**

  - Abstract: The vision of a worldwide computing network of services that Service Oriented Computing paradigm and its most popular materialization, namely Web Service technologies, promote is a victim of its own success. As the number of publicly available services grows, discovering proper services is similar to finding a needle in a haystack. Different approaches aim at making discovery more accurate and even automatic. However they impose heavy modifications over current Web Service infrastructures and require developers to invest much effort into publishing and describing their services and needs. So far, the acceptance of this paradigm is mainly limited by the high costs associated with connecting service providers and consumers. This paper presents WSQBE+, an approach to make Web Service publication and discovery

easier. WSQBE+ combines open standards and popular best practices for using external Web services with text-mining and machine learning techniques. We describe our approach and empirically evaluate it in terms of retrieval effectiveness and processing time, by using a data-set of 391 public services.

- "Empirically Assessing the Impact of Dependency Injection on the Development of Web Service Applications". Marco Crasso, Cristian Mateos, Alejandro Zunino, Marcelo Campo. *Journal of Web Engineering*, ISSN 1540-9589, vol. 9, nro.1, pp. 66-94, Rinton Press, Paramus, NJ, USA, 2010. **(Indexed by SCI)**

  – Abstract: Service-Oriented Computing (SOC) has been broadly conceived as the next big thing in distributed software development. The software industry has embraced SOC through Web Services –functionality that is accessible via ubiquitous protocols such as HTTP–. This technology provides the basis for reuse and interoperability of applications across the WWW. However, consuming Web Services is still an expensive task in terms of development costs, since developers still have to invest much effort not only into manually discovering services, but also on providing code to invoke them, which leads to software that is polluted with service-aware code and therefore is more difficult to modify and test. Recently, a technique that has become very popular for building software is Dependency Injection (DI), which allows applications to be far more testable and maintainable. In this paper, we quantitatively analyze some of the benefits and costs of DI for building Web Service applications. We base our experiments on a refined version of DI that combines text-mining, machine learning, and best practices from component-based software development to simplify the way Web Services are discovered and consumed. To our knowledge, this is the first study on the impacts of using DI in the context of SOC.

- "Improving Web Service Descriptions for effective service discovery". Juan Manuel Rodriguez, Marco Crasso, Alejandro Zunino, Marcelo Campo. *Science of computer programming*, ISSN 0167-6423, in press, Elsevier Science, Amsterdam, The Netherlands, Accepted 2010. **(Indexed by SCI)**

  – Abstract: Service-Oriented Computing (SOC) is a new paradigm that replaces the traditional way to develop distributed software with a combination of discovery, engagement and reuse of third-party services. Web Service technologies are currently the most adopted alternative for implementing the SOC paradigm. However, Web Service discovery presents many challenges that, in the end, hinder service reuse. This paper reports frequent practices present in a corpus of public services that attempt against the discoverability of any service. In addition, we have studied how to solve the discoverability problems that these bad practices cause. Accordingly, this paper presents a novel catalog of eight Web Service discoverability anti-patterns. We conducted a comparative analysis of the retrieval effectiveness of three discovery systems by using the original corpus of Web Services versus their corrected version. This experiment shows that the removal of the identified anti-patterns eases the discovery

process by allowing the employed discovery systems to rank more relevant services before non-relevant ones, with the same queries. Moreover, we conducted a survey to collect the opinions from 26 individuals about whether the improved descriptions are more intelligible than the original ones. This experiment provides more evidence of the importance of correcting the observed problems.

- "EasySOC: making Web Service outsourcing easier". Marco Crasso, Cristian Mateos, Alejandro Zunino, Marcelo Campo. *Information Sciences*, ISSN 0020-0255, in press, Elsevier Science, Amsterdam, The Netherlands, Accepted 2010. **(Indexed by SCI)**

  - Abstract: Service-Oriented Computing has been widely recognized as a revolutionary paradigm for software development. Despite the important benefits this paradigm provides, current approaches for service-enabling applications still lead to high costs for outsourcing services with regard to two phases of the software life cycle. During the implementation phase, developers have to invest much effort into manually discovering services and then providing code to invoke them. Mostly, the outcome of the second task is software containing service-aware code, therefore it is more difficult to modify and to test during the maintenance phase. This paper describes EasySOC, an approach that aims to decrease the costs of creating and maintaining service-oriented applications. EasySOC combines text mining, machine learning, and best practices from component-based software development to allow developers to quickly discover and non-invasively invoke services. We evaluated the performance of the EasySOC discovery mechanism using 391 services. In addition, through a case study, we conducted a comparative analysis of the software technical quality achieved by employing EasySOC versus not using it.

- "Combining document classification and ontology alignment for semantically enriching Web Services". Marco Crasso, Alejandro Zunino, Marcelo Campo. *New Generation Computing*, ISSN 0288-3635, in press, Springer New York, USA, Co-publication with Ohmsha, Ltd., Tokyo, Japan, Accepted 2010. **(Indexed by SCI)**

  - Abstract: Semantic Web Services represent the basic blocks for building a network of distributed and heterogeneous applications, without human intervention. Despite the high level of automatism that can be achieved with Semantic Web Services technology, this is not broadly adopted. One factor that hinders the widespread usage of this technology is the effort required to annotate semantically ordinary services. This paper presents AWSA (Automatic Web Service Annotator), an approach for easing the conversion of Web Services into Semantic Web enabled services. The main idea behind AWSA is to annotate Web Services with concepts defined by existing ontologies, which have been used for annotating similar services in the past. This approach combines text preprocessing, document classification and ontology alignment techniques to extract valuable information conveyed in standard service descriptions, reduce the search space and find proper concepts for the service being annotated, respectively. Experimental evaluations show the feasibility of the proposed approach.

6.5.2 Conferences

- "Query by example for Web Services". Marco Crasso, Alejandro Zunino, Marcelo Campo. *Proceedings of the 2008 Web Technology Track (WT) - ACM Symposium on Applied computing (SAC)*, ISBN 978-1-59593-753-7, pp. 2376-2380, Fortaleza, Ceara, Brazil, 2008.

  - Abstract: Web services have acquired enormous popularity among software developers and researchers due to the increasing levels of flexibility required by current distributed applications. However, service search facilities are still rather difficult to use. This paper presents WSQBE, a search method that aims at assisting service discoverers by generating a short list of candidate services and easing query specification. In contrast with previous approaches, WSQBE discovery process is based on a novel search space reduction mechanism. Experimental evaluations of our approach are also reported.

- "Discoverability anti-patterns: frequent ways of making undiscoverable Web Service descriptions". Juan Manuel Rodriguez, Marco Crasso, Alejandro Zunino, Marcelo Campo. *Proceedings of the 10th Argentine Symposium on Software Engineering Simposio Argentino de Ingenieria de Software (ASSE2009) - 38th JAIIO*, ISSN 1850-2792, pp 1-15, Mar del Plata, Buenos Aires, Argentina, 2009.

  - Abstract: The ever increasing number of publicly available Web Services makes standard-compliant service registries one of the essential tools to service-oriented application developers. Previous works have shown that the the descriptiveness of published service descriptions is important from the point of view of the algorithms that support service discovery using this kind of registries as well as human developers, who have the final word on which discovered service is more appropriate. This paper presents a catalog of frequent bad practices in the creation of Web Service descriptions that attempt against their chances of being discovered, along with novel practical solutions to them. Additionally, the paper presents empirical evaluations that corroborated the benefits of the proposed solutions. These anti-patterns will help service publishers avoid common discoverability problems and improve existing service descriptions.

- "Separation of Concerns in Service-Oriented Applications Based on Pervasive Design Patterns". Cristian Mateos, Marco Crasso, Alejandro Zunino, Marcelo Campo. *Proceedings of the 2010 Web Technology Track (WT) - ACM Symposium on Applied computing (SAC)*, ISBN 978-1-60558-638-0, pp. 2509-2513, Sierre, Switzerland. 2010.

  - Abstract: Service-Oriented Computing (SOC) allows developers to build applications by reusing and invoking Web-accessible services. SOC promotes loose coupling between applications and services, which has been mostly addressed by using techniques for Separation of Concerns (SoC). Contemporary SOC development models

based on SoC either rely on difficult-to-adopt, ad-hoc programming facilities and languages or fail at isolating applications from details of the application-service interaction. We propose DI4WS, a SOC programming model that combines the well-known Adapter and Dependency Injection patterns. We show that DI4WS allows reducing couplings to services, which has a positive effect on application maintenance, without requiring developers to learn such facilities or languages. DI4WS follows a contract-last approach to service invocation, whereby developers first code the logic of their applications and then non-invasively "adapt" and "inject" required services. An empirical comparison of DI4WS with two related approaches to decouple services is also reported, showing that the DI4WS versions of 4 evaluated applications used less memory and ran faster than the others.

At the same time, other articles that are not directly related to the problems discussed in this thesis have been published as well.

6.5.3 Journals & Lecture Notes

- "Supporting Ontology-Based Semantic Matching of Web Services in MoviLog". Cristian Mateos, Marco Crasso, Alejandro Zunino, Marcelo Campo. *LECTURE NOTES IN COMPUTER SCIENCE*, ISSN 0302-9743, vol. 4140, pp. 390-399, Springer, Heidelberg, Germany, 2006. **(LNCS)**

- "JEETuningExpert: A software assistant for improving Java Enterprise Edition application performance". Marco Crasso, Alejandro Zunino, Leonardo Moreno, Marcelo Campo. *Expert Systems with Applications*, ISSN 0957-4174, vol. 36, tomo 9, pp. 11718-11729, Elsevier Science, Amsterdam, The Netherlands, 2009. **(Indexed by SCI)**

6.5.4 Conferences

- "Adding Semantic Web Services Matching and Discovery Support to the MoviLog Platform". Cristian Mateos, Marco Crasso, Alejandro Zunino, Marcelo Campo. *Proceedings of the IFIP 19th World Computer Congress, TC 12: IFIP AI 2006 Stream*, ISSN 1571-5736, pp. 51-60, Santiago, Chile, 2006.

- "An Expert System for Correcting JEE Performance Anti-Patterns". Marco Crasso, Alejandro Zunino, Marcelo Campo and Leonardo Moreno. *Proceedings of the 10th Argentine Symposium on Computing Technology (AST2009) - 38th JAIIO*, ISSN 1850-2806, pp 16-30, Mar del Plata, Buenos Aires, Argentina, 2009.

Bibliography

E. Agichtein, E. Brill, S. Dumais, and R. Ragno. Learning user interaction models for predicting web search result preferences. In *29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–10, 2006.

E. Al-Masri and Q. H. Mahmoud. Qos-based discovery and ranking of Web Services. In *International Conference on Computer Communications and Networks*, pages 529–534, Los Alamitos, CA, USA, 2007.

E. Al-Masri and Q. H. Mahmoud. Discovering Web Services in search engines. *IEEE Internet Computing*, 12(3):74–77, 2008.

R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

M. Bichler and K.-J. Lin. Service-oriented computing. *Computer*, 39(3):99–101, 2006. ISSN 0018-9162.

A. Birukou, E. Blanzieri, V. D'Andrea, P. Giorgini, and N. Kokash. Improving Web Service discovery with usage data. *IEEE Software*, 24(6):47–54, 2007.

B. Blake, D. Kahan, and M. Nowlan. Context-aware agents for user-oriented Web Services discovery and execution. *Distributed and Parallel Databases*, 21(1):39–58, 2007.

M. B. Blake and M. F. Nowlan. Taming Web Services from the wild. *IEEE Internet Computing*, 12(5):62–69, 2008. ISSN 1089-7801.

P. Bollmann. The normalized recall and related measures. In *Proceedings of the 6th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 122–128, 1983.

N. Brown and C. Kindel. *Distributed Component Object Model protocol: DCOM/1.0*. Microsoft Corporation, 1998.

W. J. Brown, R. C. Malveau, H. W. McCormick, and T. J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley, 1998. ISBN 0-471-19713-0.

C. Buckley, G. Salton, and J. Allan. The effect of adding relevance information in a relevance feedback environment. In *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, 1994.

R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616, 2009. ISSN 0167-739X.

J. Cardoso and A. Sheth. Semantic e-workflow composition. *Journal of Intelligent Information Systems*, 21(3):191–225, 2003.

L. Cavallaro and E. Di Nitto. An approach to adapt service requests to actual service interfaces. In *2008 International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'08), Leipzig, Germany*, pages 129–136, New York, NY, USA, 2008. ACM Press. ISBN 978-1-60558-037-1.

Y. Chen, L. Zhou, and D. Zhang. Ontology-supported Web Service composition: An approach to service-oriented knowledge management in corporate services. *Journal of Database Management*, 17(1):67–84, 2006.

M. A. Cibrán, B. Verheecke, W. Vanderperren, D. Suvée, and V. Jonckers. Aspect-oriented programming for dynamic Web Service selection, integration and management. *World Wide Web*, 10(3):211–242, 2007.

M. Crasso, A. Zunino, and M. Campo. AWSC: An approach to Web Service classification based on machine learning techniques. *Inteligencia Artificial, Revista Iberoamericana de IA*, 37(12):25–36, 2008a. ISSN 1137-3601.

M. Crasso, A. Zunino, and M. Campo. Easy Web Service discovery: a Query-by-Example approach. *Science of Computer Programming*, 71(2):144–164, 2008b. ISSN 0167-6423.

M. Crasso, A. Zunino, and M. Campo. Combining query-by-example and query expansion for simplifying Web Service discovery. *Information Systems Frontiers*, in press, 2009. ISSN 1387-3326.

M. Crasso, C. Mateos, A. Zunino, and M. Campo. Easysoc: Making Web Service outsourcing easier. *Information Science*, in press, 2010a. ISSN 0020-0255.

M. Crasso, A. Zunino, and M. Campo. Combining document classification and ontology alignment for semantically enriching web services. *New Generation Computing*, in press, 2010b. ISSN 0288-3635.

J. de Bruijn, H. Lausen, A. Polleres, and D. Fensel. The Web Service modeling language WSML: An overview. In *ESWC*, volume 4011 of *Lecture Notes in Computer Science*, pages 590–604. Springer, 2006.

F. DeRemer and H. H. Kron. Programming-in-the-large versus programming-in-the-small. *IEEE Transactions on Software Engineering*, 2(2):80–86, 1976. ISSN 0098-5589.

X. Dong, A. Y. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for Web Services. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases*, pages 372–383, Toronto, Canada, Aug. 31 - Sept. 3 2004. Morgan Kaufmann.

N. Dourdas, X. Zhu, N. Maiden, S. Jones, and K. Zachos. Discovering remote software services that satisfy requirements: Patterns for query reformulation. *Advanced Information Systems Engineering (Lecture Notes in Computer Science)*, 4001:239–254, 2006. ISSN 0302-9743.

M. Duftler, N. Mukhi, A. Slominski, and S. Weerawarana. Web Services Invocation Framework (WSIF). In *Workshop on Object-Oriented Web Services (OOWS '01), ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '01), Tampa, Florida, USA*, 2001.

Z. Duo, L. Zi, , and X. Bin. Web service annotation using ontology mapping. In *Service-Oriented System Engineering, 2005. SOSE 2005. IEEE International Workshop*, pages 235–242, 2005.

J. Erickson and K. Siau. Web Service, Service-Oriented Computing, and Service-Oriented Architecture: Separating hype from reality. *Journal of Database Management*, 19(3):42–54, 2008.

T. Erl. *SOA Principles of Service design*. Prentice Hall, 2007. ISBN 0-13-234482-3.

P. Eugster. Uniform proxies for Java. In *OOPSLA '06: Proceedings of the* 21^{st} *annual ACM SIGPLAN conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 139–152, New York, NY, USA, 2006. ACM. ISBN 1-59593-348-4.

J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

J. Fan and S. Kambhampati. A snapshot of public Web Services. *SIGMOD Rec.*, 34(1): 24–32, 2005. ISSN 0163-5808.

C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. The MIT Press, Cambridge, Massachusetts, 1989.

A. Flores and M. Polo. Testing-based process for evaluating component replaceability. *Electron. Notes Theor. Comput. Sci.*, 236:101–115, 2009. ISSN 1571-0661.

E. Frank, M. A. Hall, G. Holmes, R. Kirkby, and B. Pfahringer. WEKA - A machine learning workbench for data mining. In *The Data Mining and Knowledge Discovery Handbook*, pages 1305–1314. Springer, 2005. ISBN 0-387-24435-2.

J. D. Garofalakis, Y. Panagis, E. Sakkopoulos, and A. K. Tsakalidis. Contemporary Web Service Discovery Mechanisms. *Journal of Web Engineering*, 5(3):265–290, 2006.

P. Gotthelf, A. Zunino, and M. Campo. A Peer-To-Peer communication infrastructure for groupware applications. *International Journal of Cooperative Information Systems*, 17(4): 523–554, 2008.

P. Grefen. Towards dynamic interorganizational business process management. In *WETICE '06: Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 13–20, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2623-3.

E. Hatcher and O. Gospodnetic. *Lucene in Action (In Action series)*. Manning Publications Co, 2004.

M. A. Hearst. Untangling text data mining. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 3–10, Morristown, NJ, USA, 1999. Association for Computational Linguistics. ISBN 1-55860-609-3.

G. T. Heineman and W. T. Councill. *Component-Based Software Engineering: Putting the Pieces Together (ACM Press)*. Addison-Wesley Professional, June 2001. ISBN 0201704854.

M. Henning. API design matters. *Commun. ACM*, 52(5):46–56, 2009. ISSN 0001-0782.

A. Heß, E. Johnston, and N. Kushmerick. ASSAM: A tool for semi-automatically annotating semantic Web Services. In *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science (LNCS)*, pages 320–334, Hiroshima, Japan, Nov. 7-11 2004. Springer. ISBN 3-540-23798-4.

M. N. Huhns and M. P. Singh. Service-Oriented Computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005.

O. Hummel, W. Janjic, and C. Atkinson. Code conjurer: Pulling reusable software out of thin air. *IEEE Software*, 25(5):45–52, 2008. ISSN 0740-7459.

W. Jian and W. Zhaohui. Similarity-based Web Service matchmaking. In *IEEE International Conference on Services Computing*, volume 1, pages 287–294, Orlando, Florida, USA, July 11-15 2005. IEEE Computer Society.

T. Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *International Conference on Machine Learning*, pages 143–151, July 8-12 1997.

R. Johnson. J2EE Development Frameworks. *Computer*, 38(1):107–110, 2005.

M. C. Juan Manuel Rodriguez, A. Zunino, and M. Campo. Improving Web Service descriptions for effective service discovery. *Science of Computer Programming*, in press, 2010. ISSN 0167-6423.

M. B. Juric, A. Sasa, B. Brumen, and I. Rozman. WSDL and UDDI extensions for version support in Web Services. *Journal of Systems and Software*, 82(8):1326–1343, 2009. ISSN 0164-1212.

T. Kawamura, T. Hasegawa, A. Ohsuga, M. Paolucci, and K. Sycara. Web Services lookup: A matchmaker experiment. *IT Professional*, 07(2):36–41, 2005.

M.-C. Kim and K.-S. Choi. A comparison of collocation-based similarity measures in query expansion. *Information Processing & Management*, 35(1):19–30, 1999.

R. I. Kittredge. Sublanguages. *American Journal of Computational Linguistics*, 8(2):79–84, 1982.

N. Kiyavitskaya, N. Zeni, J. R. Cordy, L. Mich, and J. Mylopoulos. Cerno: Light-weight tool support for semantic annotation of textual documents. *Data Knowl. Eng.*, 68(12): 1470–1492, 2009. ISSN 0169-023X.

N. Kokash. A comparison of Web Service interface similarity measures. In *3rd European Starting AI Researcher Symposium*, pages 220–231, Riva del Garda, Italy, Aug. 28-29 2006. IOS Press. Local-chair Loris Penserini.

N. Kokash, W.-J. van den Heuvel, and V. D'Andrea. Leveraging Web Services discovery with customizable hybrid matching. In *International Conference on Service-Oriented Computing*, volume 4294 of *Lecture Notes in Computer Science*, pages 522–528, Chicago, IL, USA, Dec. 4-7 2006. Springer Verlag.

R. R. Korfhage. *Information Storage and Retrieval*. John Wiley & Sons, 1997.

A. Kozlenkov, G. Spanoudakis, A. Zisman, V. Fasoulas, and F. S. Cid. Architecture-driven service discovery for service centric systems. *International Journal of Web Services Research*, 4(2):82–113, Apr.-June 2007.

D. Kramer. Api documentation from source code comments: a case study of javadoc. In *SIGDOC '99: Proceedings of the 17th annual international conference on Computer documentation*, pages 147–153, New York, NY, USA, 1999. ACM. ISBN 1-58113-072-4.

C. W. Krueger. Software reuse. *ACM Computing Surveys*, 24(2):131–183, 1992. ISSN 0360-0300.

K.-H. Lee, M.-Y. Lee, Y.-Y. Hwang, and K.-C. Lee. A framework for XML Web Services retrieval with ranking. In *International Conference on Multimedia and Ubiquitous Engineering*, pages 773–778, Seoul, Korea, Apr. 26-28 2007. IEEE Computer Society. Conference Chair Seoksoo Kim.

D. D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In *10th European Conference on Machine Learning*, volume 1398 of *LNCS*, pages 4–15, Chemnitz, Germany, Apr. 21-23 1998. Springer. ISBN 3-540-64417-2.

K. Li, K. Verma, R. Mulye, R. Rabbani, J. Miller, and A. P. Sheth. Designing semantic web processes: The WSDL-S approach. In *Semantic Web Services, Processes and Applications*, pages 161–193, 2006.

R. Losee. Sublanguage terms: Dictionaries, usage, and automatic classification. *Journal of the American Society for Information Science*, 46(7):519–529, 1995.

C. Makris, Y. Panagis, E. Sakkopoulos, and A. Tsakalidis. Efficient and adaptive discovery techniques of Web Services handling large data sets. *Journal of Systems and Software*, 79 (4):480–495, 2006.

D. Martin, M. Burstein, D. Mcdermott, S. Mcilraith, M. Paolucci, K. Sycara, D. L. Mcguinness, E. Sirin, and N. Srinivasan. Bringing semantics to Web Services with owl-s. *World Wide Web*, 10(3):243–277, 2007.

J. P. Martin-Flatin and W. Löwe. Special issue on recent advances in web services. *World Wide Web*, 10(3):205–209, 2007. ISSN 1386-145X.

C. Mateos, M. Crasso, A. Zunino, and M. Campo. Supporting ontology-based semantic matching of web services in movilog. In J. S. Sichman, H. Coelho, and S. O. Rezende, editors, *IBERAMIA-SBIA*, volume 4140 of *Lecture Notes in Computer Science*, pages 390–399. Springer, 2006. ISBN 3-540-45462-4.

A. K. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. `http://www.cs.cmu.edu/~mccallum/bow`, 1996.

S. McConnell. *Software Estimation: Demystifying the Black Art*. Microsoft Corporation, Redmond, USA, 2006.

R. McCool. Rethinking the Semantic Web, part I. *IEEE Internet Computing*, 9(6):88, 86–87, 2005.

R. McCool. Rethinking the Semantic Web, part II. *IEEE Internet Computing*, 10(1):96, 93–95, 2006.

N. Oldham, C. Thomas, A. P. Sheth, and K. Verma. METEOR-S Web Service Annotation Framework with Machine Learning Classification. In *SWSWPC*, volume 3387 of *Lecture Notes in Computer Science*, pages 137–146. Springer, 2004. ISBN 3-540-24328-3.

S. Overhage and P. Thomas. Ws-specification: Specifying Web Services using uddi improvements. In *Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*, volume 2593/2003, pages 100–119, London, UK, Feb. 2003. Springer-Verlag.

M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Semantic matching of Web Services capabilities. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 333–347, London, UK, 2002. Springer-Verlag.

M. Paolucci, K. P. Sycara, T. Nishimura, and N. Srinivasan. Using DAML-S for P2P discovery. In *IWCS*, pages 203–207. CSREA Press, 2003.

M. P. Papazoglou and W.-J. Heuvel. Service Oriented Architectures: Approaches, Technologies and Research Issues. *The International Journal on Very Large Data Bases*, 16(3): 389–415, 2007.

M. P. Papazoglou and W.-J. V. D. Heuvel. Service-oriented design and development methodology. *Int. J. Web Eng. Technol.*, 2(4):412–442, 2006. ISSN 1476-1289.

J. Pasley. Avoid xml schema wildcards for Web Service interfaces. *Internet Computing, IEEE*, 10(3):72–79, May-June 2006. ISSN 1089-7801.

A. A. Patil, S. A. Oundhakar, A. P. Sheth, and K. Verma. METEOR-S Web Service annotation framework. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 553–562, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-844-X.

C. Platzer and S. Dustdar. A vector space search engine for Web Services. In 3^{rd} *European Conference on Web Services*, pages 62–71. IEEE Computer Society, Nov. 2005.

A. L. Pope. *The CORBA Reference Guide: Understanding the Common Object Request Broker Architecture*. Addison-Wesley, Boston, MA, USA, 1998. ISBN 0-20163-386-8.

M. Porter. An algorithm for suffix stripping. *Readings in information retrieval*, pages 313–316, 1997.

S. Ran. A model for Web Service discovery with QoS. *SIGecom Exchanges*, 4(1):1–10, 2003.

J. W. Reed, Y. Jiao, T. E. Potok, B. A. Klump, M. T. Elmore, and A. R. Hurson. TF-ICF: A new term weighting scheme for clustering dynamic data streams. In *ICMLA '06: Proceedings of the 5th International Conference on Machine Learning and Applications*, pages 258–263, Washington, DC, USA, 2006. IEEE Computer Society.

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*, 1(1):77 – 106, 2005.

M. Sabou and J. Pan. Towards semantically enhanced Web Service repositories. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):142–150, 2007.

M. Sabou, C. Wroe, C. A. Goble, and H. Stuckenschmidt. Learning domain ontologies for semantic Web service descriptions. *Journal of Web Semantics*, 3(4):340–365, 2005.

H. Sagan. *Space-Filling Curves*. Springer-Verlag, New York, NY, USA, 1994.

G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

G. Salton, A.Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

B. Sapkota, L. Vasiliu, I. Toma, D. Roman, and C. Bussler. Peer-to-Peer technology usage in Web Service discovery and matchmaking. In *Proceedings of the 6th International Conference on Web Information Systems Engineering (WISE)*, pages 418–425, 2005.

C. Schmidt and M. Parashar. A Peer-to-Peer approach to Web Service discovery. *World Wide Web*, 7(2):211–229, 2004.

M. Shamsfard and A. A. Barforoush. Learning ontologies from natural language texts. *International Journal of Human-Computer Studies*, 60:17–63, 2004.

M. Shapiro. Structure and encapsulation in distributed systems: the Proxy Principle. In *Proceedings 6^{th} International Conference on Distributed Computing Systems*, pages 198–204, Cambridge, MA, USA, 1986.

K. Sivashanmugam, K. Verma, A. P. Sheth, and J. A. Miller. Adding semantics to Web Services standards. In *The 2003 International Conference on Web Services*, pages 395–401, Las Vegas, NV, USA, Sept. 2003. CSREA Press.

H. Song, D. Cheng, A. Messer, and S. Kalasapur. Web Service discovery using general-purpose search engines. In *IEEE International Conference on Web Services (ICWS)*, pages 265–271, July 2007.

D. Spinellis. The way we program. *IEEE Software*, 25(4):89–91, 2008.

R. Steinmetz and K. Wehrle. *Peer-to-Peer Systems and Applications*. Lecture Notes in Computer Science. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable Peer-to-Peer lookup service for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, Feb. 2003.

E. Stroulia and Y. Wang. Structural and semantic matching for assessing Web Service similarity. *International Journal of Cooperative Information Systems*, 14(4):407–438, June 2005.

I. Sun Microsystems. *Java Remote Method Invocation Specification*. Sun Microsystems, Inc., December 1999.

I. Toma, K. Iqbal, M. Moran, D. Roman, T. Strang, and D. Fensel. An evaluation of discovery approaches in Grid and Web Services environments. In *Proceedings of the 2nd International Conference on Grid Services Engineering and Management*, volume 69 of *Lecture Notes in Informatics*, pages 233–247, Erfurt, Thuringia, Germany, September 19-22 2005. Bonner Köllen Verlag.

M. Turner, D. Budgen, and P. Brereton. Turning software into a service. *Computer*, 36(10): 38–44, 2003. ISSN 0018-9162.

S. J. Vaughan-Nichols. Web Services: Beyond the hype. *Computer*, 35(2):18–21, Feb. 2002.

K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. METEOR-S WSDI: A scalable Peer-to-Peer infrastructure of registries for semantic publication and discovery of Web Services. *Information Technology and Management*, 6(1):17–39, 2005.

P. Vitharana, H. Jain, and F. Zahedi. Strategy-based design of reusable business components. *IEEE Transactions on Systems, Man, and Cybernetics*, 34(4):460–474, Nov. 2004. ISSN 1094-6977.

H. Wang, J. Huang, Y. Qu, and J. Xie. Web Services: problems and future directions. *Journal of Web Semantics*, 1(3):309–320, 2004.

S. Wang, L. Zhang, and N. Ma. A quantitative measurement for reputation of Web Service and providers based on cloud model. In *International Conference on Computational Intelligence for Modelling, Control and Automation*, pages 500–505, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

Y. Wang and E. Stroulia. Flexible interface matching for Web Service discovery. In *WISE '03: Proceedings of the Fourth International Conference on Web Information Systems Engineering*, page 147, Washington, DC, USA, 2003. IEEE Computer Society.

Y. Wang and J. Vassileva. A review on trust and reputation for Web Service selection. *International Transactions on Systems Science and Applications*, 3(2):118–132, 2007.

R. Witte, Q. Li, Y. Zhang, and J. Rilling. Text mining and software engineering: An integrated source code and document analysis approach. *IET Software Journal*, 2:3–16, 2008.

M. Wooldridge and N. Jennings. Software agents. *IEE Review*, pages 17–20, Jan. 1996.

B. Yang and H. Garcia-Molina. Comparing hybrid Peer-to-Peer systems. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 561–570, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

E. Yourdon and L. L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1979. ISBN 0138544719.

C. Zhou, L.-T. Chia, and B.-S. Lee. QoS-aware and federated enhancement for UDDI. *International Journal of Web Services Research*, 1(2):58–85, 2004.

H. Zhuge and J. Liu. Flexible retrieval of Web Services. *Journal of Systems and Software*, 70 (1-2):107–116, 2004.