République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider Biskra

N° d'ordre :.....

Série :.....

Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'Informatique

THÈSE

Présentée pour obtenir le grade de

DOCTORAT EN SCIENCES EN INFORMATIQUE

Par

Okba TIBERMACINE

THEME

Orchestration de Web Services Fiables

Reliable Web Services Orchestration

Soutenue le : *../../2015*

Devant le jury composé de :

M. Noureddine DJEDDI	Professeur à l'Université de Biskra	Président
M. Foudil CHERIF	Maître de conférences A à l'Université de Biskra	Rapporteur
M. Chouki TIBERMACINE	Maître de conférences à l'université de Montpellier, France	Co-Rapporteur
M. Abdelouahab MOUSSAOUI	Professeur à l'université de Setif 1	Examinateur
M. Salah SADOU	Maître de conférences HDR, à l'université de Bretagne Sud, France	Examinateur
M. Abdelmalik BACHIR	Maître de conférences A à l'Université de Biskra	Examinateur

Reliable Web Services Orchestration

Abstract:

Service Oriented Computing (SOC) represents a paradigm for building distributed applications over the Internet. Service Oriented Architecture (SOA) is an architectural style that allows developing these applications based on services. Over the past decade, Web services orchestration has become thriving area of research and academic efforts. Although many orchestration-related challenges have been tackled, orchestration reliability and its verification are still an open, critical and prerequisite issue as web service orchestration are affecting several life-days activities.

This thesis addresses the subject of reliable web service orchestrations. In particular, it contributes a set of approaches, techniques and tools to address the selection and orchestration of reliable web services. Firstly, it refines life-cycle phases of service orchestration to ensure reliability analysis and verification during orchestration design and run-time. As well as it proposes an architecture, based on an enhanced service registry, to achieve the same goal. Secondly, it introduces an approach for measuring web services similarity based on their interfaces that serves as a basic technique for identifying similarity, substitutability and composability relationships between web services. Besides, a Tool coined WSSIM, implementing this approach is developed, and it is available for reproducible research. The tool is experimented on a set real web services for validation. Thirdly, the thesis contributes with an approach for the identification of simple and complex service substitutes. The latter is built upon similarity measurement, FCA classification and reliability analysis. In addition, a set of algorithms are presented to describe the functionality of different phases in the identification process. Fourthly, to consider reputation of services as a 'softer' form or reliability, the thesis introduces a framework and a mathematical model for web service reputation management.

Keywords: Service oriented architecture, Web services selection and recommendation, Similarity assessment, Reputation management, Reliability verification and analysis.

Orchestration de Web Services Fiables

Resumé: L'Informatique Orienté Services représente un paradigme pour construire des applications distribuées sur Internet. L'Architecture Orientée Services(SOA) est un style architectural qui permet le développement de ces applications à base de services. Au cours de la dernière décennie, l'orchestration des services Web est devenue un domaine très actif dans la recherche scientifique et académique. Bien que de nombreux défis liés à l'orchestration aient été abordés, la fiabilité de l'orchestration et de sa vérification restent encore un sujet ouvert, prérequis et important de fait que ces orchestrations affectent aujourd'hui plusieurs activités quotidiennes.

Cette thèse focalise sur le sujet d'orchestration des Services Web Fiables. En particulier, elle contribue avec un ensemble d'approches, de techniques et d'outils pour améliorer la sélection et l'orchestration des services web fiables. Premièrement, elle affine les phases du cycle de vie d'orchestration de services web afin d'assurer une vérification continuée de fiabilité lors des phases de conception et d'exécution. En outre, elle propose une architecture conceptuelle basée sur un registre de service amélioré, pour la mise en ÅŞuvre d'orchestrations fiables. Deuxièmement, elle présente une approche de mesure de similarité entre les services web. L'approche repose sur la comparaison des interfaces WSDL de services. L'approche sert à identifié les relations de similarité, de substituabilité et de composabilité entre services. L'outil WSSIM a été développé pour mettre en oeuvre l'approche proposée. Pour validation, l'outil a été expérimenté avec un ensemble important de services web réels. Troisièmement, la thèse contribue avec une approche pour l'identification des substituts de services simples et complexes. L'approche utilise les techniques de mesure de similarité, la classification de service avec FCA et l'analyse de fiabilité pour identifier et sélectionner les meilleurs substitutes. Un ensemble d'algorithmes aient été proposés pour décrire le processus d'identification. Quatrièmement, pour examiner la réputation des services comme un autre critère de fiabilité, la thèse introduit un Framework et un modèle mathématique pour la gestion de réputation de service Web.

Keywords:

Architecture Orientée Services, Selection et Recommendation des Services Web, Measure de Similarité, Gestion de Réputation, Vérification de Fiabilité

ملخص :

تقدم الحوسبة خدماتية التوجه (Service Oriented Computing) إطارا عاما يهتم بتطوير التطبيقات الموزعة على شبكة الانترنت. و توفر البنية الخدماتية (Service Oriented Architecture) النموذج الهندسي الذي يسمح ببناء هذه التطبيقات على أساس وحدات تسمى خدمات (Services) . كما تعتبر خدمات الويب (Services Web) ، مع تنسيقها و دمجها (Web Services Orchestration and Composition) حلولا تفعيلية لهندسة البنية الخدماتية. لذلك عرفت العشرية الأخيرة اهتماما متزايدا من قبل الباحثين بدراسة هذه المواضيع نظريا و تطبيقيا. و على الرغم من كون العديد من المشاكل تم التطرق اليها و لو بصور جزئية، إلا أن موثوقية تنسيق و دمج خدمـــات الويب (Orchestration and Composition reliability) و مراقبتها على مدى دورة حياة الخدمات البسيطة او المدمجة، تبقى موضوعا حرجا و مفتوحا للدراسة، كما أنه يكتسي أهمية بالغة كون هذه الخدمات صارت تؤثر على عدة نشاطات في الحياة اليومية.

و تأتي هذه الرسالة في اطارها العام لتقدم دراسة حول دمج و تنسيق الخدمات الموثوقة، و تساهم بشكل أدق، في تقديم مجموعة من الحلول و المناهج و الأدوات لانتقاء خدمات الواب الموثوقة و دمجها و تنسيقها. إذ تقدم أولا تعديلا على مراحل دورة خدمات الويب الناتجة عن عملية التنسيق و الدمج و ذلك لمراقبة موثوقيتها. كما تقدم مقترحا هندسيا لبناء هذا النوع من البرامج اعتمادا على سجل متقدم لنشر خدمات الواب. ثانيا، تقدم منهجية لقياس التشابه بين خدمات الويب وفقا لملفات وصف واجهاتها، وهذا الذي يسمح بالكشف عن علاقات التشابة، التبادل و التركيب بين هذه الخدمات. بالاضافة إلى ذلك، تم تطوير برنامج يدعى WSSIM الذي يقوم بقياس التشابه، و هذا البرنامج مفتوح لاستعماله في البحوث العلمية المشابهة. ثالثا، تشارك الرسالة بتقديم منهجية اخرى لاكتشاف البدائل الموثوقة لخدمات الويب المعطلة. و هذه المنهجية مبنية على دراسات التشابه، و التصنيف بواسطة طريقة تحليل المفاهيم الرسمية (Formal Concept Analysis) و تحاليل الموثوقية. رابعا، للاخذ بعين الاعتبار سمعة الخدمة كمعنى من معاني الثقة، تقدم الرسالة أيضا إطارا و انموذجا رياضيا يقوم بتقييم سمعة خدمات الواب و مزوداتها بناء على آراء المستخدمين السابقين لتلك الخدمات. مما يساعد في اختيار خدمات الويب الاكثر موثوقية قبل و أثناء عملية التنسيق و الدمج.

كلمات مفتاحية : البنية الخدماتية، اختيار و اقتراح خدمات الويب، قياس التشابه، تسيير السمعة، تحاليل و مراقبة الموثوقية.

Acknowledgments

All praise and thanks to my lord, Allah !

I am grateful to Dr. Foudil Cherif and Dr. Chouki Tibermacine my academic supervisors, for their support, guidance, assistance and encouragements during this long journey.

I deeply thank Jury members: Prof. Djeddi Noureddine, Dr. Salah Sadou, Dr. Abdelmalik Bachir, and Prof. Abdelouahab Moussaoui, for accepting to evaluate this work. In addition, I would like to express my sincere respect to the anonymous reviewers of the published papers who provided me with helpful critics, suggestions and guidance.

My thanks go to my colleagues at the computer science department of Biskra university. I am also grateful for friends who helped me. I sincerely thank them all for their support.

I would like to mention my family (my mother, my wife, my son, my brothers, my sisters and all my relatives). I cannot say enough thankful to them simply because they are the reasons for me to keep on trying.

Finally, to all of my dear friends, who I cannot name all of them since I am afraid of missing someones, I sincerely appreciate their friendships, which have brought different meanings to my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Context

Nowadays, the need for advanced service-based software systems is in continuous increase, following the growth of consumer requirements that become more demanding and massively complex. This kind of software systems are both modeled and developed following service-oriented paradigm, that defines the necessary conceptual fundamentals to cope with requirements complexity and integration challenges by promoting the development of autonomous, loosely coupled, platform-independent and reusable software entities called services [127, 128].

The idea behind this paradigm is to allow organizations to offer their business functionalities as services with well defined interfaces over the Internet, and thus, other companies can use, compose, and/or integrate these services into more capable business application eventually available for others as services. This vision of developing distributed applications is depicted into an architectural style called Service Oriented Architecture (SOA) [59], that allows services to be published, discovered and consumed by applications or other services.

SOA principal stakeholders and their roles are depicted within The SOA triangle (see Figure 1.1). Interactions between these stakeholders are conducted as

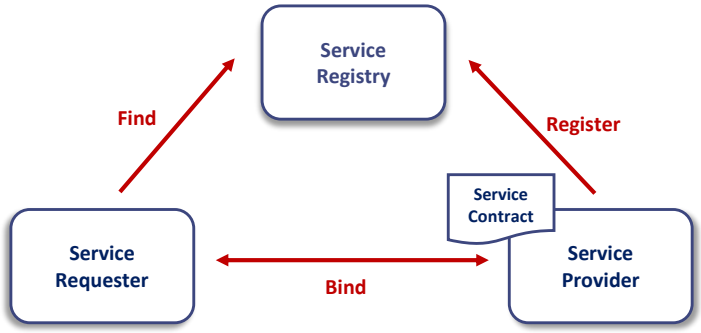

Figure 1.1: The SOA triangle

follows: First, a service provider that offers a business functionality as a service, publishes ('Registers') a service contract that describes the service registry. Second, a service requester can query the registry to 'find' specific services. Third, based on the results returned by the registry, the requester can dynamically bind to one of the service.

Practically, an implementation of SOA is achieved by using Web services standards and technologies, including (i) WSDL (Web Service Description Language [33]) that serves as a language to describe service interfaces (contracts), (ii) UDDI (Universal Description, Discovery and Integration [18]) that represents a service registry for storing and publishing services contracts, and (iii) SOAP (Simple object Access Protocol [24]) that serves as a transport protocol that allows interactions between service requesters and providers.

One of the most challenging topics in SOA and web services technology, that was extensively studied in industry and academia, is web services composition, which is known also as web service orchestration. In general, web services orchestration is the process that combines several web services to form a single web service that performs more complex functionality [146]. The result of service orchestration is called composite service [19, 28].

Although many challenges and issues related to web services orchestration have been tackled theoretically and practically in the literature, web services orchestration reliability and its verification are still an open, critical and prerequisite issue as these web services orchestrations are present in our everyday life affecting different functions and situations [73]. In general, system reliability is considered as the probability that the system completes its task whenever it is invoked [97]. Hence, for web services orchestrations users, reliability means that composite services, and thus its orchestrated services, work correctly and without interruptions, and they are available when they are required. In addi-

tion, reliability includes the satisfaction of Quality of Service (Qos) minimum requirements defined by orchestration designers. Moreover, reliability for service designers and stakeholders can be considerer as the ability to trust that services offered by other providers fulfills requirements and works as expected [73].

This thesis addresses some orchestration reliability-related problems that are precisely defined in the next sections.

1.2 Problem Statement

The problem to be addressed in this thesis can be expressed by the following research question:

How can reliable web services orchestration be ensured through the selection of reliable and reputable atomic service candidates and their substitutes at different phase of orchestration life cycle?

In particular, this main research question can be divided into the following two sub-questions:

- *How are reliable atomic Web services selected for web services orchestration at design phase?*

  Reliability of web service orchestration depends mainly on reliabilities of its atomic services [73]. Ideally, orchestration designer has to select service candidates that answers required functionalities and ensures high Quality of Service. Thus, reliability built a confidence to the technical capabilities of the selected services. Moreover, reliability can have "softer", non-technical forms that manifest by reputation and trust. (i.e., aggregation of experiences with the service or service provider) and trust (i.e., Trust is gained from person's or organization's experiences with the service or service provider). Thus, reliable service candidates have to be also reputable and trustful.

- *How do we select service substitute candidates that could replace simple or composed unreliable web services at run-time and/or maintenance phases?*

  To maintain the required reliability, the service system must be able to update itself (self-reconfiguration) or updated by the orchestration stakeholder to maintain the required reliability in case of failure of one of its atomic services, or in case of detection of undesirable changes in its reliability. That is, atomic service candidates that become unreliable have to

be replaced by reliable substitutes that guarantee the same functionalities and required reliability (QoS and reputation).

These questions raise the need for a set of methods and frameworks for effectively guarantee the selection of reliable atomic service candidates and their substitutes.

More specifically, in this thesis we address the following points:

1. Reliable web service orchestrations,
2. Similarity assessment between web services,
3. Identification of service substitutes,
4. Web service reputation measurement.

Note that related problems such run-time monitoring, dynamic reconfigurations and adaptation, etc. are out of the scope of this thesis.

1.2.1 Reliable web service orchestration

Web service reliability and its verification is prerequisite in failure-prone environments, as these services and their orchestrations are affecting several everyday on-line activities. Thus, there is a major need to develop reliable service orchestration architecture that ensure the verification of reliability alongside orchestration's [1] life cycle. This architecture has to be designed in a way that:

- Facilitates the selection of reliable web services during different life-cycle phase;
- Allows the identification, thus the recommendation, of service substitutes for better replacement and adaptation;
- Permits the conduction of continuous reliability analysis;
- Provides mechanisms for evaluating accurately the reputation of web service and the trustworthiness of their providers;
- Affords mechanisms for run-time service QoS monitoring;
- Simplifies requirement documentation for orchestration stakeholders;
- Enables to compose, manage and update service orchestrations at run time, etc.

[1] Service orchestration and service composition are often used interchangeably in this thesis

1.2.2 Web service interface similarity measurement

The study of similarity between web services is one of the underlying techniques for service classification, selection and service substitutes identification. Goals behind studying similarity between web services are multiples, including:

- Select and retrieve from registries, catalogs and portals web services that match some keywords and satisfy some criteria.
- Identify substitutes (replacements) between web services or between their operations.
- Determine service composability i.e., whether in two operations, belonging to two web services, the output of the first operation is similar to the input of the second operation. Thus, the first service can be composed with the second service.

Therefore, there is a need for similarity assessment approaches and tools that could achieve these goals. Mainly, these approaches and tools propositions have to take into account the absence of web service behavior description and to focus on the statical content description presented by Web service description language (WSDL) files. Moreover, they should address structural and semantic aspects of these descriptions during the comparison (i.e, Matchmaking) process.

1.2.3 Web service substitutes selection

Web services are software component that are exposed to errors and failures that may have different origins: a failure in the network, the unavailability of the application server or the database server, an error or an exception in the program implementing the service, etc. Some of these faults cannot be detected immediately, and it could have many consequences on service compositions using them [93]. In the literature, service substitution and adaptation is a technique proposed to overcome the failure in service orchestration by replacing defected web services. Though identifying and selecting substitutes is a crucial for orchestration healing, it is still a challenging task, especially with the absence of behavioral descriptions of web services.

Thus, there is a demand for approaches that simplify the identification and the selection of simple and complex service substitutes, based on the failed service description file and the required reliability criteria.

1.2.4 Web service reputation management

In the context of web services, reputation is seen as a collective measurement of the opinion of a community of users regarding their actual experiences with

the web services [105, 121]. Reputation reflects the reliability, trustworthiness and credibility of web services and their providers [176], which consider it as an important factor for web service selection and recommendation [143, 173, 177, 183, 188].

In fact, the measurement of web service reputation is challenging due to many factors including the presence of malicious users that could inject false feedback ratings into the reputation system. Thus, pure feedback ratings have to be distinguished from malicious ratings. In addition, user credibility has to be fairly evaluated and other factors like age of ratings have to be considered during reputation assessment.

Hence, there is a need for a reputation management framework that faces these challenges and evaluates accurately the reputation and trustworthiness or web services and their providers, in order to conduct more successful reliable service selection and recommendation.

1.3 Contributions and main results

To address some of the challenges described previously, this thesis makes five major contributions: (i) an architecture for reliable web service orchestrations, (ii) a practical approach for the measurement of similarity between web services, (iii) a process for the identification of reliable web service substitutes, (iv) a framework and a mathematical model for the management of web service reputation, and (v) a reputation bootstrapping technique for estimating initial reputation of newcomer services .

- ⊛ **Architecture for reliable Web service orchestrations**: In which, we propose a general architecture that assists stakeholders to ensure reliable web service orchestrations. The architecture is built upon three main components: (i) service provider, (II) orchestration stakeholder, and (III) a web service recommendation system that enhances classical web service registries and catalogs. In this architecture, the third component serves as a central unit that ensures the recommendation of reliable web services with a continuous reliability analysis, and a notification service for orchestration subscribers. Moreover, the component ensures other tasks such as:
  - Registers services, providers and clients,
  - Monitors on-demand web services and tracks their QoS,
  - Assesses similarity between services,

- Identifies service substitutes and evaluates their reliability,
- Evaluates reputation scores of services and their providers,
- Notifies providers and clients with updates and occurred changes, etc.

⊛ **Practical approach for the measurement of web service similarity** [C,D] [164,165]: In which we propose a solution to assess the similarity between web services by analyzing their interfaces; The approach compares between elements in the two matched WSDL files (e.g., operations, messages, parameters ...etc) and determines the similarity score between them. Similarity scores range between 0 and 1, where 0 means a total dissimilarity and 1 means a complete similarity. The approach uses different lexical and semantic similarity metrics to evaluate how close elements are to each other. Moreover, the measurement process is parametrized by a collection of weights associated to the different levels of web service description. The challenge of measuring the similarity between complex types, which are generally represented by XML schema, is handled by using different techniques (e.g., similarity flooding) for getting accurate scores.

A tool coined 'WSSIM' that implements this approach is developed using Java programming language and a set of other underlying technologies. The tool is publicly available[2] for reproducible research. The approach and its implementing tool have been experimented for validation on a set of real-world Web services.

⊛ **A Process of relevant substitutes identification for healing failed Web service orchestrations** [B] [166] To ensure the reliability in web service orchestration, reliable service substitutes have to be identified before and during orchestration runtime. Therefore, even though a failure occurs in the orchestration, the system will be able to replace the defected service and thus to ensure its continuous functionality under equivalent or better QoS. For this end, we propose an approach for identifying relevant web service substitutes. The approach is built upon two techniques; the first is the similarity assessment between web services that enable to evaluate similarity and composability relationships between a set of web service candidates; and the second technique is the Formal concept analysis (FCA), which is used for classifying the compared services to retrieve substitutes.

⊛ **Reputation management framework for Web services** In which we propose an architecture with an assessment model for reputation management of web services and their providers. The approach aims to accurately assess the reputation of web services based on user feedback ratings.

[2]WSSIM is available at https://code.google.com/p/wssim/

The proposed framework provides some solutions to the deficiencies and limitations found on the literature. The contribution of this framework can be summarized as follows:

- First, we propose a generic architecture for the management of web service reputation to facilitate web service selection and recommendation. We describe the main components that handle feedback rating acquisition, storing and aggregation, in addition to service recommendation.
- Second, we propose an enhanced reputation assessment model that encloses rating time sensitivity and user credibility factors. The first factor permits to assign more weights to new feedback ratings. We believe that the behavior of service providers is in continuous change due to the dynamic nature of their environment. Therefore, an effective reputation assessment model should assign more importance to recent feedback ratings.

  The second factor ensures the evaluation of user's honesty which has its impact on considering fair feedback ratings. We apply a user penalization mechanism to exclude unfair ratings received from users who do not guarantee a certain credibility (i.e., less or equal than a threshold).
- We propose a formula for the evaluation of the overall reputation of web service orchestration for better recommendation.
- Finally, we conduct experiments for evaluating the performance and accuracy of the proposed reputation assessment model.

⊛ **Regression-Based Bootstrapping technique of Web Service Reputation Measurement** [A] [167] in which we introduce a bootstrapping model for evaluating the initial reputation of web services newly published. The technique is based on three main phases:

- Provider reputation evaluation: The system assesses the reputation of the new service provider from the reputation of its previously published web services.
- Reputation estimation from similar web services: First, the system selects among the long-standing web services those which are similar to the newcomer service. Second, the system builds an equation model based on: i) reputation scores and ii) QoS values of similar services. Finally, the system, based on the established model, estimates a reputation value of the newcomer service.

– Regression-based reputation estimation: Likewise to the second phase, the system builds a multiple linear regression model from QoS and reputation data of all long-standing web services. This model enables the estimation of the unknown reputation value of newcomer services from their known values of QoS.

The final reputation is assigned to the newcomer web service depending on the results of the three phases.

1.4 Thesis outline

This thesis contains a background and state-of-the-art study, a description of the proposed contributions, and some conclusions and perspectives. This content is organized in 8 chapters as follows:

⁕ **Chapter 2** presents a background material on: Web Services fundamentals (Sect. 2.2), some similarity measurement schemes and metrics used to evaluate the similarity between texts and documents (Sect. 2.3), the Formal Concept Analysis technique (FCA) for objects classification (Sect. 2.4). As well as basic concepts of reputation and trusts (Sect. 2.5).

⁕ **Chapter3** studies the related works, draws advantages and limitations of existing approaches, and spotlights challenges and opening issues of thesis related subjects.

⁕ **Chapter 4** describes the different phases of reliable web services orchestration's life cycle (Sect. 4.2). Moreover, the chapter introduces an architecture for reliable web service orchestration (Sect. 4.3).

⁕ **Chapter 5** describes the proposed Web services similarity measurement approach (Sect. 5.2). Then, it presents the tool WSSIM that implements the approach (Sect. 5.3), as well as it describes the conducted experiments and validation process (Sect. 5.4).

⁕ **Chapter 6** introduces an approach for the identification of service substitutes. The chapter describes the illustrative example used alongside the description of identification process (Sect. 6.2). Second, it covers the description of the architecture and the components of the approach (Sect.

6.3). Third, it shows the conducted experiments and validation methodology and obtained results (Sect. 6.2).

⁎ **Chapter 7** describes contributions four and five. It presents a framework for web service reputation management. The chapter starts by describing the architecture and its core components (Sect. 7.2). It presents the mathematical model proposed for the assessment of service and provider reputations (Sect. 7.3). Then, it introduces the bootstrapping model and its evaluation (Sect. 7.4). Finally, it presents the conducted experiments for validating the proposed reputation assessment model (Sect. 7.5).

⁎ **Chapter 8** concludes the thesis with a summary of contributions (Sect. 8.1), and outlines some future research directions (Sect. 8.2).

Related publications

A- Okba Tibermacine, Chouki Tibermacine, Foudil Cherif, Regression-Based Bootstrapping of Web Service Reputation, to appear in Proceedings of the 13th IEEE International Conference on Web Services (ICWS'15), Application Track, New York, USA, June-July 2015. IEEE Computer Society.

B- Okba Tibermacine, Chouki Tibermacine, Foudil Cherif, A process to identify relevant substitutes for healing failed WS-* orchestrations, Journal of Systems and Software, Elsevier, Volume 104, June 2015, Pages 1-16, ISSN 0164-1212.

C- Okba Tibermacine, Chouki Tibermacine and Foudil Cherif. A Practical Approach to the Measurement of Similarity between WSDL-based Web Services. In (RNTI) (French-Speaking Journal: Revue des Nouvelles Technologies de l'Information), special issue of selected papers from CAL 2013, volume RNTI-L-7, 2014, Pages 3-18.

D- Okba Tibermacine, Chouki Tibermacine and Foudil Cherif. WSSim: a Tool for the Measurement of Web Service Interface Similarity. In proceedings of the french-speaking conference on Software Architectures (CAL'13), Toulouse, France. May 2013.

It is important to note that all relevant publications co-authored by the author of this thesis are referenced in this chapter. The thesis mainly summarizes these publications and in the remaining chapters their content is used without being referenced individually.

Part I

State of the Art

Chapter 2

Background

Contents

2.1 Overview

This chapter describes some of the background information needed to facilitate the understanding of the research work described in the second part of this thesis. Section 2.2 starts by giving a brief survey of the most important aspects of Web services and Web service orchestrations. Section 2.3 shows some Information Retrieval (IR) techniques used for measuring the similarity between structured documents. These techniques are applied for evaluating the similarity between web service interfaces as it is described in Chapter 5. In addition, Section 2.4 presents a classification technique called Formal Concept Analysis that we use in our substitute identification process for classification of web services and their substitutes. Moreover, Section 2.5 summarizes concepts and notions of reputation and trust in the web service context.

2.2 Web service fundamentals

2.2.1 Web Services

Web services have many definitions that range from the very generic to the very specific. For instance, a web service is considered as an application accessible to other application over the web (see e.g., [30]). This definition is very generic and any resource with a universal Resource Locator (URL) can be seen as web service.

A second definition provided by IBM is that " *Web services are a new breed of web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the web. Web services perform functions, which can be anything from simple requests to complicated business processes. Once a web service is deployed, other applications (and other web services) can discover and invoke the deployed service*" [112]. This formal definition gives more details and stresses that web services should be open, that is, they need to be published, located and invoked over the web.

In addition, another formal definition of web services is given by the World Wide Web Consortium [1] (W3C) as follows :" *a software application identified by a Universal Resource Identifier (URI), whose public interfaces and bindings are defined, and described using eXtensible Markup Language (XML). Its definition*

[1] World Wide Web Consortium: www.w3c.org

can be discovered by other software systems. These systems may then interact with the web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols" [174]. This definition emphasizes that web services should be able to be defined, described, discovered and clearly mentions that XML is a part of the solution to allow web services to be integrated into more complex distributed application.

Moreover, Papazoglou [125] summarizes the existing definitions into the following : "*a Web service is a platform-independent, loosely coupled, self-contained, programmable Web-enabled application that can be described, published, discovered, coordinated, and configured using XML artifacts (open standards) for the purpose of developing distributed interoperable applications*".

There are two main categories of web services: the traditional SOAP-based web services and the RESTful web services. The first category are also called WS-* web services, their development depends on three important standardization initiatives, i.e., WSDL, SOAP, UDDI. The registration, discovery and invocation of these services are implemented by SOAP calls. The second category of web services are based on the Representational State Transfer (REST) model [63] which was introduced as an architectural style for building large-scale distributed hypermedia systems. RESTful web services identified by URIs, and interact through a uniform interface, i.e., a fixed set of operations of the Hypertext Transfer Protocol (HTTP): GET, PUT,DELETE and POST.

This thesis focuses only on the study of SOAP-based web services which are used typically to integrate complex enterprise applications.

2.2.2 Web service model and its underlying technologies

In this subsection, we present the web service model, and a set of standards used for describe, publish, discover and invoke web services.

Web service model

The basic web service model (see Figure 2.1) is set up of three types of participants (web service roles) including : *service provider*, *service registry* and *service client* (requester). Interactions between these three participants are: service *publishing*, *finding* and *binding* (invoking). These interactions depend on Web service artifacts, which include *service description* and *service implementation* [192].

Participants in the Web service model are described as follows:

- *Service provider* is the owner of the web service. The owner holds the

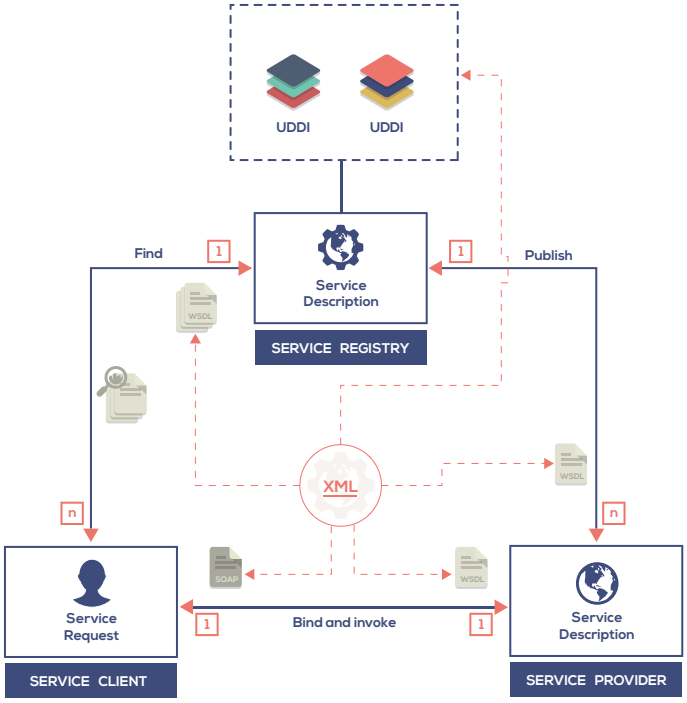

Figure 2.1: W3C Web service reference model

implementation of the service application and makes it accessible via the web (i.e., identified by an URI).

- *Service client* represents a human or a software agent that intends to make use of some services to achieve a certain goal.
- *Service registry* (or service broker) is a searchable registry providing service descriptions. It implements a set of mechanisms to facilitate service providers to publish their service descriptions. Meanwhile, it also enables service clients to locate services and get binding informations.

Interactions are organized in three modes:

- *Service publication* is to make the service description available in the registry so that the service client can find it.
- *Service lookup* is to query the registry for a certain type of service and then retrieve the service description.
- *Service binding* is to locate, contact, and invoke the service based on the binding information in the service description.

Artifacts includes the service implementation and description:

- Service implementation is a network accessible software module realized by the service provider. It could be invoked by a service client or act as a service client to interact with another service provider.
- Service description could contain the syntactic and semantic information of the Web services. The syntactic information describes the input/output of the operations, the binding information, the data types, and so on.

The next subsection presents the different technologies used for the concretization of the web service reference model, including WSDL, SOAP and UDDI. These technologies are a subpart of the Web service technology stack.

Web Service Description Language

Legacy web services are described using the Web Service Description Language (WSDL) [33]. The later is an XML-based language that contains several elements to describe functionalities of a web service. Logically, a WSDL document can be seen as two parts; an abstract and a concrete part. The abstract part describes operations of the web service that form its interface, along with their input and output parameters and the XML schema definition that specifies parameter data types. The concrete part describes how the service should be bound, that is, it describes the binding protocol and service's endpoints.

A typical WSDL document has a root definition element <wsdl:definitions>, which comprises the following sections:

- Service section (<wsdl:service>) specifies the service name and a collection of concrete implementation endpoints of Web service.
- Port describes how a binding is deployed at a particular network endpoint.
- Binding section (<wsdl:service>) describes the implementation details of how the elements in a portType are converted into a concrete implementation in a particular combination of data formats and protocols.
- PortType section (<wsdl:portType>) is a Web service's abstract interface definition where each child operation element defines an abstract method signature.
- Messages section (<message>) defines the format of the message, or a set of input or output parameters, referred to by the method signatures or operations. A message is composed of parts.
- Types section (<wsdl:types>) defines the collection of all the data types used in the Web service as referenced by the various message part elements.

Simple Object Access Protocol

Simple Object Access Protocol (SOAP) [24] is an XML-based language that defines a flexible binding framework for exchanging structural messages for invocation and result response between web services. SOAP provides a message construct that can be exchanged over a variety of underlying protocols (such as HTTP/HTTPS TCP, UDP, BEEP, SMTP, or JMS).

A SOAP message is an ordinary XML document, consisting of a SOAP envelop, which is the root element of the message, with an optional header and a SOAP body.The header contains information that indicate how the message is routed to reach its final destination, and how it can be preprocessed and by whom. The body is a container for mandatory information intended for the ultimate recipient of the message.

Universal Description, Discovery and Integration

Universal Description, Discovery and Integration (UDDI) [18] is an XML-based standard that offers a mechanism to classify, catalog and manage web services, so that they can be discovered, located and consumed [125].

UDDI provides two basic specifications that define a service registry's structure and operation [44]:

- A definition of the information to provide about each service, and how to encode it.
- A query and update API for the registry that describes how this information can be accessed and updated.

The UDDI business registry consists of three directories:

- **Business Entity** (or white pages), which lists the providers and their related basic information such as a company name, address, and phone numbers, as well as other standard business identifiers like tax numbers.
- **Business Service** (or yellow pages), which provide a classification of services based on standard taxonomies.
- **Binding Template** (or green pages), which provides information about a company's key business processes, such as operating platform, supported programs, purchasing methods, shipping and billing requirements, and other higher-level business protocols.

It is important to note that UDDI's are not the unique place to search for web services; service discovery can be conducted also through web service portals and search engines such as ProgrammableWeb [2], WebServiceList [3] and Xmethods[4].

2.2.3 Quality of Service

Quality of Service (QoS) is a set of characteristics (properties) that describes the non-functional aspects (quality aspect) of a service. QoS refers to the ability of the service to respond to expected invocations and to perform them at a level that corresponds to the mutual expectations of both its provider and its consumer [125]. Truong *et al.* [168] presented a detailed taxonomy of web services QoS (see figure 2.2). By evaluating the QoS aspects of a set of Web services that share the same goals, a consumer could identify which service meets the quality requirements of the request. QoS attributes are influenced by the Internet's dynamic and unpredictable nature. Therefore, delivering good QoS and selecting services with Good QoS are critical and significant challenges [8].

In the following we present a list of QoS parameters for web services:

- Performance. Represents the speed in which a service request can be completed, measured in terms of throughput, response time, execution time, latency and transaction time [69, 154]:
  - Throughput. Number of Web service requests served within a period of time [154].
  - Response time. Time consumed between invocation and completion of the requested service operation [45, 69].
  - Processing time (execution time). Time taken by a Web service to process a request [69].
  - Latency. Time consumed between the service request arrives and the moment it is served [83].
  - Transaction time. Time used by the service to complete a transaction [69].
- Availability. Probability that the system is ready to be used. The service should be available when it is invoked [69].

[2] http://www.programmableweb.com
[3] http://www.webservicelist.com
[4] http://www.xmethods.com

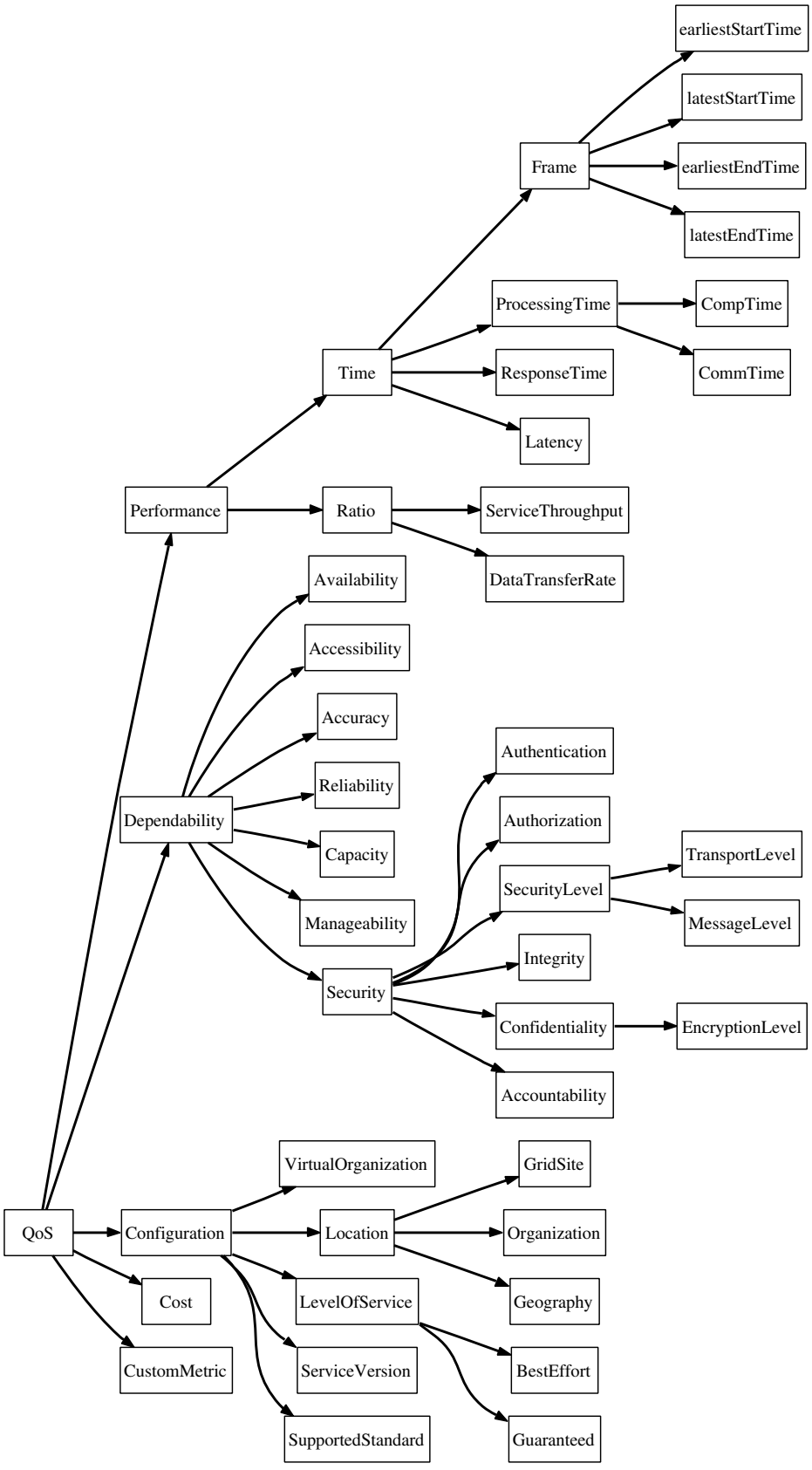

Figure 2.2: QoS parameters [168]

- Reliability. Probability that the request is correctly responded, maintaining the service quality [45]. A measure of reliability can be the number of failures per period of time (day, week, etc.) [69, 154].
- Accessibility. Property of a service to serve a request from a consumer [154].
- Security. Ability to ensure authorization, confidentiality, traceability/auditability, data encryption, and non-repudiation [69].
- Scalability. Ability of increasing the computing capacity of a service provider's computer system to process more requests, operations or transactions in a given period of time [69].

2.2.4 Web services composition

The true capacity of SOA and web service technologies is achieved through composing multiple services into more capable and powerful applications [146]. Web service composition is performed by aggregating either atomic (also called elementary [144]) or composite services. A composite service [19, 28] is an umbrella structure that brought together other composite and atomic services that collaborate to implement a set of operations (Or a business process, i.e, a set of logically related tasks performed to achieve a well-defined business outcome [125]). A well known example of composite service is the travel preparation service (e.g., [192]), which integrates services for booking flights, booking hotels, searching for attractions, etc.

Service composition can be realized either by service orchestration or service choreography [129]. Orchestration is where a central or master element (service) controls all aspects of the process (see Figure 2.3 (a)). Choreography is where each element of the process is autonomous and controls its own behavior (see Figure 2.3 (b)).

Web services orchestration

Web services orchestration represents a single executable business process that coordinates the interaction among the different services, by describing a flow from the perspective and under control of a single endpoint [125]. A service orchestration includes management of the transactions between these services, including any necessary error handling, as well as describing the overall process [146]. In a services orchestration, invoked Web services neither know and nor need to know that they are being orchestrated as part of the high level service (i.e., the composite service) and that they are playing a role in a business process definition. Only the central coordinator of the orchestration is aware of this goal,

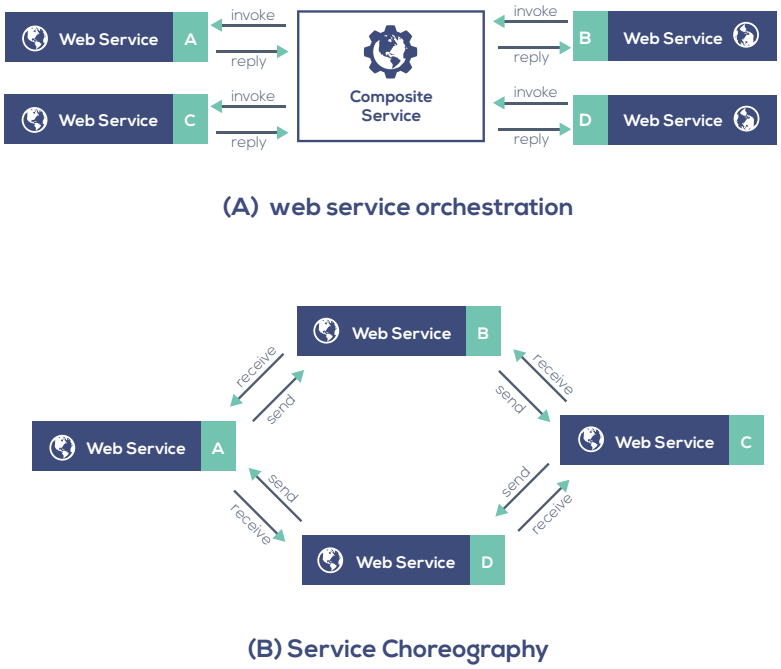

Figure 2.3: Service orchestration and service choreography

so the orchestration is centralized with explicit definitions of operations and the order of invocation of web services [81].

Standards for web service orchestration include the Business Process Modeling Notation (PBMN) [179] and the Business Process Execution Language (WS-BPEL or BPEL in short) [4, 22]. The former is used for defining the visual representation of the sequence, and BPEL represents the code that executes the sequence. The two languages are largely supported by the industry.

Web services choreography

Service choreography is non-executable abstract process that defines the interaction protocol between the involved web services from a global perspective. That is, each partner describes its part in the interaction, which is defined by public exchange of messages, rules of interaction and agreements between two or more business process endpoints [146]. Choreography is typically associated with the interactions that occur between multiple web services rather than specific business process that a single party execute like in service orchestration.

The standard that support web service choreographies is the Web Service Choreography Description Language [84] (WS-CDL).

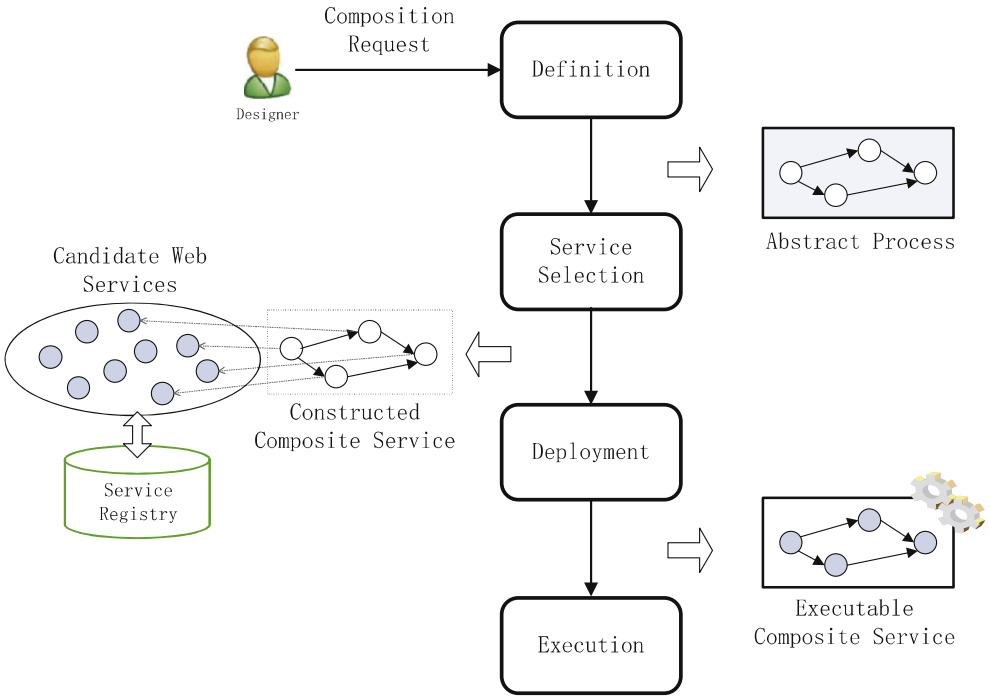

Figure 2.4: Service composition life cycle [146]

2.2.5 Web services composition life cycle

A service composition life cycle is divided into four phases, including definition, service selection, deployment and execution phases. Figure 2.4 schematizes the service composition life cycle. These phases are described according to Sheng *et al.* [146] as follows:

– **Definition phase**: in this phase, the designer specifies the services composition requirements which are then decomposed, either semi-automatically or automatically, into an abstract process. The abstract process specifies a set of activities, the control and data flow among them, the Quality of Service (Qos) requirements, and the exceptional behavior.

– **Service selection phase**: in this phase, for each activity in the business process, relevant web services that match activity's requirements are located by searching services registries or service search engines. Web services search is conducted based on information contained in the published service description document. Often, more then one service candidate will meet these requirements. Therefore, the best matched service with the highest Qos are selected. Once the required web services are identified and

bound to the corresponding activities, the constructed composite service is produced.

– **Deployment phase**: The constructed composite service is deployed on servers, allowing its instantiation and invocation by end users. The result os this phase is an executable composite service.

– **Execution phase**: in this phase, instances of the composite service are created and executed by the execution engines, which is responsible for invoking the individual service components. During composite service run time, monitoring tasks, including logging, execution tracking, performance measuring and exception handling, should be performed [146].

2.2.6 Web service composition languages

In this subsection, we present some standard languages used in academia and industry for web service composition.

Web Services Business Process Execution Language

Web service Business Process Execution (WS-BPEL) [4], or BPEL in short, is an XML-based language for Web services orchestration. In BPEL, the orchestration result is called a process, participating services are called partners, and message exchange or intermediate result transformation is called an activity.

A partner represents either a consumer service of the implemented business process, or a provider of a service used in the business process. That is, it can be a web service that the process invokes, or any client that invokes the process. The BPEL interacts with each partner using a **partnerLink** construct. The partner link establishes a conversation channel between the process and the partner service.

BPEL introduces several types of primitive activities to:

– allow for interaction with the applications being composed (e.g., **invoke**, **reply**, and **receive** activities).

– wait for some time (the **wait** activity).

– copy data from one place to another (the **assign** activity).

– indicate error conditions (the **throw** activity).

– terminate the entire composition instance (the **exit**).

– do nothing (the **empty** activity).

These primitive activities can be combined into more complex ones using structured activities (also called constructs) provided by BPEL such as **sequence**, **while**, **If** and **flow**.

Moreover, a particular construct offered by BPEL is scope, which provides a way to divide a complex business process into hierarchically organized parts. In a scope, there is a collection of activities that can have their own variables, fault handlers, and compensation handlers.

A fault handler gets executed when an exception arises, for example through the execution of the throw activity, while compensation handlers are triggered due to faults or through compensate activities that force compensation of a scope.

An interesting feature of BPEL is its support for two distinct styles of process modeling: the graph-oriented style, involving definition of a composition using graph primitives (nodes and edges), and the 'algebraic' style derived from process calculi, in which complex control constructs (such as the compound activities above) result in implicit control flow [146]. Each of these alone provides sufficient expressibility. Supporting both styles gives the designer maximum flexibility to develop a model in the most intuitive way.

Web Services Choreography Description Language

Web Services Choreography Description Language (WS-CDL) [84] is an XML-based language that describes peer-to-peer collaborations of participants by defining their observable behavior. WS-CDL captures service interactions from a global perspective, meaning that all participating services are treated equally, which is different from BPEL where service interactions are described from one single participant perspective. WS-CDL supports exception handling through a special kind of workunit, namely Exception, which is associated with a choreography and is enabled when an exception occurs. Another special kind of workunit is Finalizer, which is enabled when its associated choreography completes successfully. A finalizer can confirm, cancel or modify the effects of completed actions of the choreography, which can be used to provide a compensation mechanism, as well as a range of coordination models.

Business Process Modeling Language

Business Process Modeling Language, or BPML in short, is an XML-based language that is proposed by the Business Process Management Initiative[5]. Initially developed to describe business processes that can be executed by a business process management system. The later version of BPML incorporates many concepts of Web Service Choreography Interface (WSCI), which focuses on the

[5] (BPMI.org)

choreography of Web services. BPML provides basic and structural activities that are similar to those in BPEL. Basic activities are used to invoke the available services (action), assign a new value to a message (assign) and instantiate a process (call). Structured activities are used to manage the branch selection (choice and switch), repetition (until and while), sequential (sequence) and concurrent activities (all). BPML allows developers to schedule tasks to determine the time to perform the task by using schedule.

A context is important element in BPML. It contains local definitions that specify common behavior for all activities within the same context, which can be used to exchange information and coordinate execution. Signal is used to synchronize between parallel activities executing within the same context. Designed for long-running business processes, BPML also supports the feature of persistency. Nested processes could be established by aggregating several subprocesses. Two transaction mechanisms, atomic and open nested transactions, are provided in BPML. The former is for short-lived transactions while the latter is for long-running transactions. BPML also provides the exception handling mechanism (exception process) to deal with exceptional events, and the compensation mechanism (compensation process) to reverse the effects of a completed activity [146].

Electronic Business Using XML

Electronic Business Using XML or ebXML in short, aims at defining a set of specifications for enabling business-to-business (B2B) interactions among companies of different size. ebXML consists of the following components: (1) Messaging service provides a standard way to exchange business messages between organizations. The messaging service does not rely on any particular file transport mechanism (such as SMTP, HTTP, or FTP) or network for exchanging data. (2) Registry is a database of items that support doing business electronically. This component stores important information about businesses such as XML schemas of business documents, definitions of library components for business process modeling, and trading partner agreements [146].

(3) Trading partner information. The Collaboration Protocol Profile (CPP) provides the definition of an XML document that specifies the details of how an organization is able to conduct business electronically. The Collaboration Protocol Agreement (CPA) specifies the details of how two organizations have agreed to conduct business electronically. (4) Business Process Specification Schema (BPSS) provides the definition of an XML document modeling business processes (i.e., composite services). It identifies the overall business process, the roles, transactions, identification of the used business documents (the DTDs or

schemas), document flow, legal aspects, security aspects, business level acknowledgments, and status.

OWL-S

OWL-S previously DAML-S (DARPA Agent Markup Language for Web Services), is a language that provides the ability to describe and reasoning semantically over services. This language comprises three ontologies: (1) service profile, (2) process model, and (3) grounding ontology. The service profile ontology is used for describing services to facilitate the service discovery. Service descriptions and queries are built from a description of functional properties (e.g., inputs, outputs, and preconditions) and non-functional properties (e.g., QoS parameters). Moreover, the service profile class can be specialized to create profile taxonomies that subsequently describe different classes of services. Besides, Process models describe the composition and execution of Web services. The process model is used both for reasoning about possible compositions (e.g., validation) and for controlling the enactment and invocation of a service. OWL-S defines three possible process classes: composite, simple, and atomic.

Atomic processes are directly invocable and have no subprocesses. Simple processes are not invocable and provide a means of describing service or process abstractions. A simple process does not have any specific binding to a physical service and thus has to be realized either by an atomic process, or expanded into a composite process. Composite processes are hierarchically defined workflows, consisting of atomic, simple and other composite processes. Finally, the grounding of a service specifies the details of how to access the service. The process model is mapped to a WSDL description of the service, through a thin grounding. Each atomic process is mapped to a WSDL operation, and the OWL-S properties used to represent inputs and outputs are grounded in terms of XML data types. Additional properties pertaining to the binding of the service are also provided (e.g., the IP address of the machine hosting the service, the ports used to expose the service) [146].

2.3 Similarity measurement schemes

Similarity measures are essential for pattern recognition problems such as clustering, classification and information retrieval problems [96]. This section presents some similarity metrics and techniques that we employ in the assessment of similarity between web service interfaces (for details, see similarity assessment approach in Chapter 5).

2.3.1 Vector space model

To reduce the analysis complexity of textual documents and to make them easier to handle, document has to be transformed from the full text version to document vector which describes the content of the text document. One of the used technique for such transformation is the vector space model (VSM). Vector space model [140] is an algebraic model that represents text document as a vector of identifiers, such as, index terms.

The procedure of VSM can be divided to three phases: (i) document indexing where content bearing terms are extracted from the full text document. (ii) weighting of the indexed terms to enhance retrieval content of document relevant to the user. (iii) ranking documents with respect to user's query according to a similarity measure.

With a VSM, both documents and queries are represented as vectors :

$$d_j = \{w_{1,j}, w_{2,j}, ..., w_{t,j}\}$$

$$q_j = \{w_{1,q}, w_{2,q}, ..., w_{n,q}\}$$

Where, d_i is the document number i in the corpus (collection) and $w_{i,j}, 1 \leq i \leq t$ is the words (terms) of this document, and q_j in the user query number j. If a term occurs in the document, its value in the vector is non-zero. Different way of computing these values, also known as (term) weights, are proposed in the literature, including Term Frequency -Inverse Document Frequency (TF-IDF) for instance.

2.3.2 Term Frequency - Inverse Document Frequency

Term Frequency - Inverse Document Frequency or (TF-IDF for short) is a numeric statistic technique that is used for scoring and term indexing. TF-IDF evolved from IDF [150] that comes with the idea that a term in a query which occurs in many documents is not a good discriminator, and should be given less weight than one which occurs in few documents [196]. besides, TF measures how frequently a term occurs in a document.

The basic formula of TF-IDF for term weighting is the following :

$$w_{i,j} = tf_{i,j} \times \log(\frac{N}{df_i})$$

where

- $w_{i,j}$ is the weight for the ith term in the jth document.
- N is the number of document in the collection (corpus).

- $tf_{i,j}$ is the term frequency of the ith term in the jth document.
- df_i is the document frequency of the ith term in the collection.

Example

A simplified example of the vector space model with TF-IDF is presented in this subsection (The example is taken from [74]). Consider a very small collection C that consists in the following three documents:

d_1:" new york times"

d_2:" new york post"

d_3:" los angeles times"

The total number of documents is $N = 3$, the idf values for the term are:

new $\log_2(3/2) = 0.584$

york $\log_2(3/2) = 0.584$

times $\log_2(3/2) = 0.584$

post $\log_2(3/1) = 1.584$

los $\log_2(3/1) = 1.584$

angeles $\log_2(3/1) = 1.584$

For all the document, we calculate the tf scores for all the terms in C. We assume the words in the vector are ordered alphabetically.

	angeles	los	new	post	times	york
d_1	0	0	1	0	1	1
d_2	0	0	1	1	0	1
d_3	1	1	0	0	1	0

Then, we multiply the tf scores by the idf values for each term.

	angeles	los	new	post	times	york
d_1	0	0	0.584	0	0.584	0.584
d_2	0	0	0.584	1.584	0	0.584
d_3	1.584	1.584	0	0	0.584	0

Given the following query "new york times", we calculate the $tf - idf$ vector for the query, and compute the score of each document in C in relative to this query, using the cosine similarity measure. When computing the $tf - id$ values for the query terms we divide the frequency by the maximum frequency (2) and multiply with the idf values.

q 0 0 (2/2)*0.584=584 0 (1/2)*0.584=0.292 0

We calculate the length of each document and of the query

Length of d_1 = sqrt(0.584^2+0.584^2+0.584^2)=1.011

Length of d_2 = sqrt(0.584^2+1.584^2+0.584^2)=1.786

Length of d_3 = sqrt(1.584^2+1.584^2+0.584^2)=2.316

Length of q = sqrt(0.584^2+0.292^2)=0.652

Then, the similarity values are :

cosSim(d_1,q) = (0∗0+0∗0+0.584∗0.584+0∗0+0.584∗0.292+0.584∗0)/(1.011∗0.652) = 0.776

cosSim(d_2,q) = (0*0+0*0+0.584*0.584+1.584*0+0*0.292+0.584*0) / (1.786*0.652) = 0.292

cosSim(d_3,q) = (1.584*0+1.584*0+0*0.584+0*0+0.584*0.292+0*0) / (2.316*0.652) = 0.112

According to the similarity values, the order in which the documents are returned as result to the query will be :d_1,d_2,d_3.

2.3.3 Latent Semantic Indexing

Latent Semantic Indexing (LSI) is an extension of the vector space method [49]. LSI can retrieve relevant documents even when they do not share any words with the query. Therefore if only a synonym of the keyword is present in a document, the document will be still found relevant. The idea behind LSI is to transform the matrix of documents by terms in a more concentrated matrix by reducing the dimension of the vector space. The number of dimensions becomes much lower, there is no longer a dimension for each term, but rather a dimension for each 'latent' concept or group of synonyms (though it is not clear what is the desired number of concepts). The dimensionality of the matrix is reduced by a mathematical process called Singular Value Decomposition.

The advantage of LSI is its strong formal framework that can be applied for text collections in any language and the capacity to retrieve many relevant documents. But the calculation of LSI is expensive, so in practice it works only for relatively small text collections [74].

2.3.4 Similarity and Distance Measures

Terminology

Similarity: measure of how close to each other two instances are. The closer the instance are to each other, the larger is the similarity value.

Dissimilarity: measure of how different two instances are. Dissimilarity is large when instances are very different and is small when they are close.

Proximity : refers to either similarity or dissimilarity.

Distance metric : a measure of dissimilarity that obeys to the following laws (Laws of triangle norm) [163]:

- $d(x, y) \leq 0; d(x, y) = 0, \text{iff} x = y;$
- $d(x, y) = d(y, x);$
- $d(x, y) + d(y, z) \geq d(x, z).$

Conversion of similarity and dissimilarity measures [50]: if we denote $s(x, y)$ the similarity between x and y. We can revert the similarity to serve as the dissimilarity measures $(d(d, y))$ and vice versa. Thus, if we have $(d(d, y))$ we can use

$$s(x, y) = \frac{1}{d(x, y)}$$

or

$$s(x, y) = \frac{1}{d(x, y) + 0.5}$$

as the corresponding similarity measure. if similarity measure values range between 0 and 1 (so called degree of similarity), then the corresponding dissimilarity measures can be defines as

$$d(x, y) = 1 - s(x, y)$$

Euclidean Distance

The Euclidean distance between two points is given by Minikowski distance metric. It can be used in one-, two-, or higher-dimensional space. Mathematically, the Euclidean distance between two points p and q is given as follows [142]:

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2}$$

where n is the number of dimensions. The euclidean distance measures the numeral difference for each corresponding attributes of point p and q. Then, it combines the square of differences in each dimension into an overall distance.

Cosine Similarity

The cosine similarity is a measure of similarity of two non-binary vector. The typical example is the document vector, where each attribute represents the frequency with which a particular word occurs in the document. Each document vector is sparse since it has relatively few non-zero attributes. Therefore, the cosine similarity ignores 0-0 matches. The cosine similarity is defined by the following equation [160]:

$$\cos(A, B) = \frac{A \times B}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}.$$

The resulting similarity ranges from -1 meaning exactly opposite, to 1 meaning exactly the same. with 0 usually indicating independence, and in-between values indicating intermediate similarity or dissimilarity.

For text matching, the attribute vectors A and B are usually the term frequency vectors of the documents. The cosine similarity can be seen as a method of normalizing document length during comparison. In the case of information retrieval, the cosine similarity of two documents will range from 0 to 1, since the term frequencies (tf-idf weights) cannot be negative. The angle between two term frequency vectors cannot be greater than 90.

Jaccard Coefficient

Jaccard coefficient [77] and distance measures respectively the similarity and dissimilarity (diversity) between finite sample sets. This coefficient is defined as the size of the intersection between the two sets divided by the size of their union.

Mathematically, Let A and B be two finite sets. The Jaccard coefficient that measures the similarity between the two sets is defined as follows:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

If the two sets are empty, we define $J(A, B) = 1$. Clearly, $0 \leq J(A, B) \leq 1$.

The Jacard distance is defined as follows:

$$d_J(A, B) = 1 - J(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}.$$

Levenshtein Distance

Levenshtien [92] is a string metric for measuring the distance between two strings (character sequences). It is defined as the minimum number of operations to transform one string into the other, where the operations may be deletion of a character, insertion of a character or substitution of a character. For instance, the distance between the Word **take** and **cake** is one, because the character **t** in the first word must be substituted by character **c** to get the second word.

Mathematically, the levenshtien distance between string **a** and **b** is given by the function $lev_{a,b}(|i|, |j|)$, which is defined as follows:

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

where, $1_{(a_i \neq b_j)}$ is the indicator function equal to 0 when $(a_i = b_j)$ and 1 otherwise.

Jaro-Winkler Distance

Jaro-Winkler distance [181] is a similarity measurement distance between strings. It is a variant of the Jaro distance proposed in [78]. The Jaro measure is the weighted sum of percentage of matched characters. Winkler increased this measure for matching initial characters, then rescaled it. The higher the Jaro-winkler distance for two strings is, the more similar the strings are. The score is normalized such that 0 equates to no similarity and 1 is an exact match.

Mathematically [36], given strings $s = a_1...a_K$ and $t = b_1...b_L$, define a character a_i in s to be common with t there is a $b_j = a_i$ in t such that $i - H \leq j \leq i + H$, where

$$H = \frac{\min(|s|, |t|)}{2}$$

. Let $s' = a'_1...a'_K$ be the characters in s which are common with t (in the same order they appear in s) and let $t' = b'_1...b'_L$ be analogous; now define a transposition for s' and t' to be a position i such that $a'_i \neq b'_i$. Let $T_{s',t'}$ be half the number of transpositions for s' and t'. The Jaro similarity metric for s and t is

$$Jaro(s,t) = \{ \begin{array}{ll} 0 & \text{if s'=0,} \\ \frac{1}{3}\left(\frac{|s'|}{|s|} + \frac{|t'|}{|t|} + \frac{|s'| - T_{s',t'}}{|s'|}\right) & \text{otherwise.} \end{array} \}$$

The Winkler variant uses the length P of the longest common prefix of s and t. Letting $P' = max(P, 4)$ we define

$$\text{Jaro-Winkler}(s,t) = Jaro(s,t) + \frac{P'}{10}.(1 - Jaro(s,t))$$

For illustration, given the string s MARTHA and t MARHTA, we find : $|s'| = 6$, $|s| = 6$ and $|t| = 6$. There are mismatched characters T/H and H/t, thus, $T = \frac{2}{2} = 1$. The Jaro score between s and t :

$$Jaro(s,t) = \frac{1}{3}\left(\frac{6}{6} + \frac{6}{6} + \frac{6-1}{6}\right) = 0.944$$

To find the JaroWinkler score, we have $P' = 3$. Using the standard weight $(\frac{1}{10})$ we get:

$$\text{Jaro-Winkler}(s,t) = 0.944 + \frac{3}{10}.(1 - 0944) = 0.961.$$

2.4 Formal Concept Analysis

Formal Concept Analysis (FCA) [67, 180] is a technique of data mining and branch of mathematical lattice theory. FCA is used, among others, in data analysis, information retrieval, software engineering and data mining. FCA analyzes data which describes a relationship between groups of objects that share common attributes and provides an associated graphical representation.

The main goal of FCA is to model concepts of thought as a unit of two parts:

- The concept Extension which comprises all objects that belong to the concept.
- The concept Intention which contains all attributes that these objects share.

We present in this section the fundamental concepts of FCA.

2.4.1 Case study

We explains the FCA technique using a case study about some programming language classification [6]. Tables 2.1 classifies some programming languages according to the following criterion:

[6] This classification is taken from the following website: http://techdistrict.kirkk.com/2009/03/30/programming-language-classification/ (visited:01/03/2015)

Table 2.1: Programming language classification table

Language	Type System	Problem Space	Runtime Environment	Paradigm
Java	Static	General	Managed	OO, Imperative
Ruby	Dynamic	General	Managed	OO, Imperative
C	Static	General	Unmanaged	Procedural, Imperative
C++	Static	General	Unmanaged	OO, Imperative
Python	Dynamic	General	Managed	OO, Imperative
PHP	Dynamic	General	Managed	OO, Imperative
Fortran	Static	General	Unmanaged	Procedural, Imperative
Perl	Dynamic	General	Managed	Procedural, Imperative
COBOL	Static	General	Managed	Procedural, Imperative
SQL	Dynamic	DSL	Managed	Declarative
WS-BPEL	Dynamic	DSL	Managed	Declarative
Lua	Dynamic	General	Managed	Functional, Imperative
smalltalk	Dynamic	General	Unmanaged	OO, Imperative
Objective-C	Static	General	Unmanaged	OO, Imperative
ABAP	Static	General	Managed	OO, Imperative
Erlang	Dynamic	General	Managed	Functional
F#	Static	General	Managed	OO, Functional
Scala	Static	General	Managed	OO, Imperative, Functional
M	Dynamic	Purpose	Managed	Declarative
Clojure	Dynamic	General	Managed	OO, Imperative, Functional

– Type system: Indicates whether the la language is **Static** or **Dynamic**.

– Problem space: indicates if the language is **General**, or **Domain Specific**, or for a given **Purpose**.

– Run-Time environment: indicates if the language's environment is **Managed** (i.e., garbage collection, etc.) or an **Unmanaged** environment.

– Paradigm: indicates to which programming paradigm the language belongs; Object-Oriented (**OO**), **Procedural**, **Functional**, **Imperative**, or **Declarative**.

2.4.2 Formal Context

A **formal context** is denoted as $K = (G, M, I)$ where G is a set of **objects**, M is a set of **attributes**, and I is a **binary relation** between G and $M \quad (I \subset G \times M)$, $(g, m) \in I$ is read: "object g has attribute m" such that $g \in G$ and $m \in M$.

Table 2.2: A formal context of programming language classification

	Dynamic	Static	General	DSL	Purpose	Managed	Unmanaged	OO	Imperative	Declarative	Procedural	Functional
Java		x	x			x		x	x			
Ruby	x		x			x		x	x			
C		x	x				x		x		x	
C++		x	x				x	x	x			
Python	x		x			x		x	x			
PHP	x		x			x		x	x		x	
Fortran		x	x				x		x		x	
Perl	x		x			x			x		x	
COBOL		x	x			x			x		x	
SQL	x			x		x				x		
WS-BPEL	x			x		x				x		
Lua	x		x			x			x			x
smalltalk	x		x				x	x	x			
Objective-C		x	x				x	x	x			
ABAP		x	x			x		x	x			
Erlang	x		x			x						x
F#		x	x			x		x				x
Scala		x	x			x		x	x			x
M	x				x	x				x		
Clojure	x		x			x		x	x			x

A formal context is represented as a cross table in which, objects appear as row labels and attributes as column labels. A cross in the cell (g, m) of this table indicates that the object g has the attribute m [8].

From the language classification case study, we build a formal context of programming languages *G={Java, Ruby, C, C++, Python, PHP, Fortran, Perl, COBOL, SQL, WS-BPEL, Lua, Smalltalk, Objective-C, ABAP, Erlang, F#, Scala, M, Clojure}* and their related characteristics *M={Dynamic, Static, General, DSL, Purpose, Managed, Unmanaged, OO, Imperative, Declarative, Procedural, Functional}*. The corresponding formal context is depicted in Figure 2.2.

In addition, for a set $X \subseteq G$ of objects, we define the set $X' \subseteq M$ of attributes, which are common to the objects in X, as follows:

$$X' = \{m \in M | gIm, \forall g \in X\}$$

Analogously, we define for a set $Y \subseteq M$ of attributes, the set $Y' \subseteq G$ of objects, which have all the attributes in Y, as follows:

$$Y' = \{g \in G | gIm, \forall m \in Y\}$$

For the example, if we take the set X=\{*C, PHP, Fortran*\} from Table 2.1, the set of common attributes X' =(\{*C, PHP, Fortran*\})'=\{*General, Imperative, Procedural*\}.

2.4.3 Formal Concept

A **formal concept** of the context $K = (G, M, I)$ is a pair (X, Y), where $X \subseteq G$ is called the **extent**, $Y \subseteq M$ is called the **intent**, $X' = Y$ (or equivalently $Y' = X$), meaning that a concept is a maximal collection of objects sharing a maximal collection of attributes. The set of all concepts of the context K is denoted as $\mathbf{B}(G, M, I)$, The closure operator of X is defined as $X'' = X$ [8].

For instance, (\{*C, PHP, Fortran*\},\{*General, Imperative, Procedural*\}) is a formal concept. However, (\{*Objective-C*\},\{*Static, General, Unmanaged, OO, Imperative*\}) is not a concept, because (\{*Objective-C*\})'= \{*Static, General, Unmanaged, OO, Imperative*\} while (\{*Static, General, Unmanaged, OO, Imperative*\})'= \{*Objective-C, C++*\}.

2.4.4 Object and attributes concepts

For an object $g \in G$, we define its **object intent** as $g' = \{ m \in M | gIm\}$. Analogously, we define for an attribute $m \in M$ its **attribute extent** as m' =$\{g \in G | gIm\}$. Hence, an **object concept** (g'', g') is denoted as γg and the **attribute concept**(m', m'') is denoted μg.

For example, $\gamma(SQL)$ is ($(SQL)''$,$(SQL)'$)=(\{*SQL, WS-BPEL*\}, \{*Dynamic, DSL, Managed, Declarative*\}).

2.4.5 Subconcept and superconcept

If we have two concepts (X_1,Y_1) and (X_2,Y_2), we say that (X_1,Y_1) is a **subconcept** (X_2,Y_2), when $X_1 \subseteq X_2$ (equivalently $Y_1 \subseteq Y_2$). Inversely, we say that (X_2,Y_2) is a **superconcept** of (X_1,Y_1). The relation between the two concepts is (X_1,Y_1) $\leq$ (X_2,Y_2), the relation $\leq$ is the **order** relation of the concepts [8].

2.4.6 Concept lattice

The $\mathbf{B}(G, M, I)$ provided with the order relation $\leq$ is a **concept lattice** and is denoted as $\underline{\mathbf{B}}(G, M, I)$.

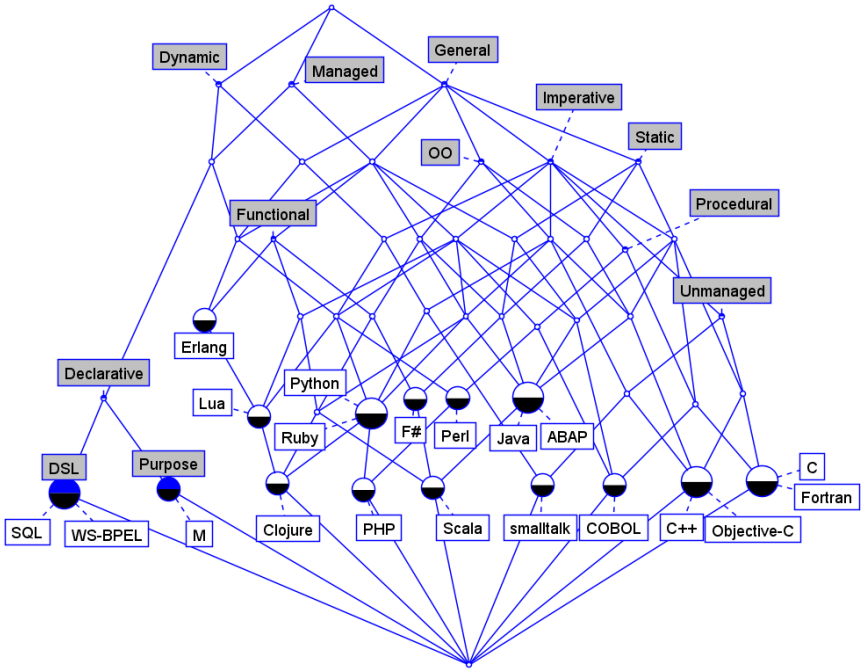

Figure 2.5: The Concept Lattice for the Context in Table 2.2

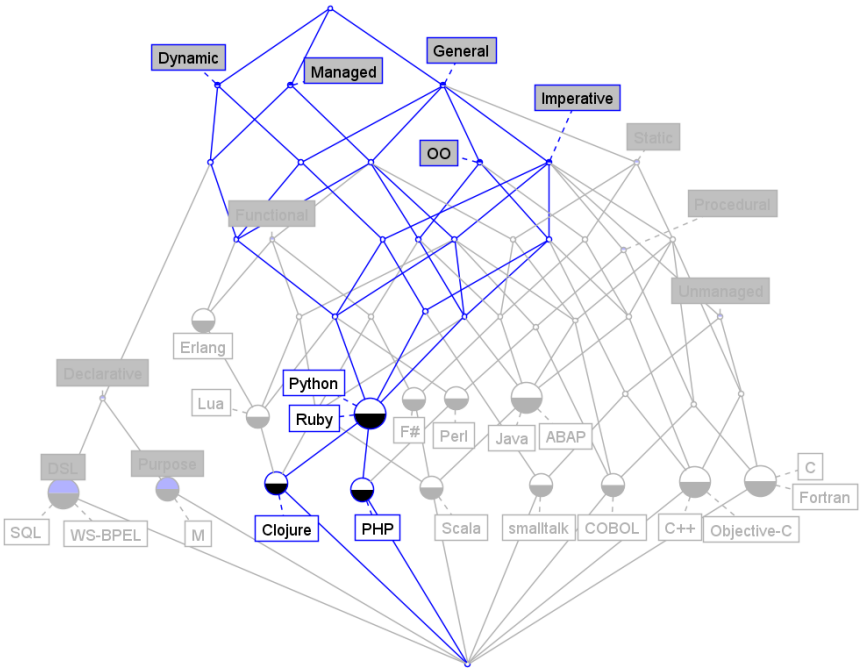

Figure 2.6: The resulted concept lattice focusing on one concept

Figure 2.5 depicts the lattice built from the context language classification context shown in Table 2.2. The lattice is generated using the concept Explorer software [189]. Figure 2.6 represents the lattice when we focus on one of the identified concepts.

2.4.7 FCA for web Services

In the context of web services, FCA has been successfully applied for web service selection and classification, because it offers a formal classification and browsing mechanism thereby allowing the organization of web services in groups that share common characteristics (e.g. similarity values, keywords, QoS attributes, operation signatures, and/or functionalities). In addition, FCA allows to visually representing this classification by concept lattices that facilitate the navigation and the browsing for needed services and their potential substitutes.

Technically, a concept in the web service context is modeled in a unit of two parts:

1. Concept extension (G) which comprises all the considered web services. In specific cases, the concept extension comprises parts of the web services such as the operations, or even more input and output messages.

2. Concept intention (M) which holds the manipulated attributes that could be keywords, operations, messages,functionalities, QoS attribute, or only meaningful strings that represent any other web service (operation/message) characteristics.

During FCA application for the classification of web services, we build the so-called square concepts. They are defined as concepts with equal intention and extension sets ([8]); i.e. these concepts form square gatherings on the binary context matrix. They allow the identification of groups of mutually related objects (web services, operations, or messages). A better recognition of square concepts is achieved by performing mutual column/line interchange in the binary context matrix. Concrete examples about square concepts are provided during the presentation of the identification process of service substitutes.

2.5 Trust and Reputation

The development of distributed software systems requires the interaction of entities (e.g., agent, services) and the use of resources from diverse organizations throughout a network. The concept of trust is crucial to ensure secure interactions, especially when these entities surpass the boundaries of a community, which has clear security preferences [3].

2.5.1 Trust

Trust is a directional relationship between two parties that can be called trustor and trustee. The trustor could be any "thinking entity" that has the ability to make assessment and decision based on received information and experience. The trustee can be any physical or abstract entity such as organization, person, information, service, etc [79].

The literature uses the term trust with a variety of meanings [37]. Two main interpretations are to view trust as the perceived reliability of something or somebody, called "reliability trust", and to view trust as a decision to enter into a situation of dependence, called "decision trust".

- **Reliability trust**: *Trust is the subjective probability by which an individual, A, expects that another individual, B, performs a given action on which its welfare depends* [80].

From this definition, trust is primarily defined as the trustor's estimate of the trustee's reliability (e.g. expressed as probability) in the context of dependence on the trustee.

- **Decision Trust** : Trust is the extent to which a given party is willing to depend on something or somebody in a given situation with a feeling of relative security, even though negative consequences are possible.

In this definition, trust is primarily defined as the willingness to rely on a given object, and specifically includes the notions of dependence on the trustee, and its reliability.

Both reliability trust and decision trust reflect a positive belief about something on which trustor depends for his welfare. Reliability trust is most naturally measured as a discrete or continuous degree of reliability, whereas decision trust is most naturally measured in terms of a binary decision

2.5.2 Reputation

The concept of reputation is related to that of trustworthiness. According to Merriam-Webster's online dictionary reputation is defined as follows:

- **Reputation** :*The overall quality or character as seen or judged by people in general.*

Reputation can be considered as a collective measure of trustworthiness (in the sense of reliability) based on the referrals or ratings from members in a community. An individual's subjective trust can be derived from a combination of received referrals and personal experience. In order to avoid dependence and loops it is required that referrals be based on first hand experience only, and not on other referrals. As a consequence, an individual should only give subjective trust referral when it is based on first hand evidence or when second hand input has been removed from its derivation base [80].

Reputation can relate to a group or to an individual. A group's reputation can for example be modeled as the average of all its members' individual reputations, or as the average of how the group is perceived as a whole by external parties. The study of Tadelis et al. [159] have shown that belonging to a given group will inherit an a priori reputation based on that group's reputation. If the group is reputable all its individual members will a priori be perceived as reputable and vice versa.

2.5.3 Trust management systems

Trust Management Systems (TMS) are classified into three categories [156]:

- **Credential-based TMS**: In these systems, service provider and the provided service are both trusted, but service requesters are not. Service provider use credentials to estimate the trustworthiness of service requesters, and based on the estimation the provider decide whether to grand to service or not to its requesters.
- **Social network-based TMS**: These systems are based on social networks. Reputation is measured based on social relationships.
- **Reputation-based TMS**: in such systems, the service provider and its service are both not trusted. Service requesters select services based on the reputation values. Reputable service providers are selected to provide requested services.

2.5.4 Reputation computation methods

Reputation of an entity is computed from users' feedback ratings. In the literature, different computation methods have been introduced, including:

- Simple summation: The total reputation score of an entity is computed as the total number of positive feedback ratings minus the total number of

negative feedback ratings. For instance eBay system [7] applies this method for reputation computation [138].

- Average ratings: The total reputation score of an entity is calculated as the average of feedback ratings. For instance, this method is used by Amazon[8] and Epinions[9] [100].

- Discrete trust models: In these models, feedback ratings are provided in verbal form (e.g., high, medium, low, good ...), because humans are often able to rate more successfully in discrete verbal measures [3].

- Bayesian systems: these systems (e.g., [55, 116, 117]) compute reputation scores by statistical updates of beta probability density function. It takes a binary input positive or negative, and computes reputation based on the previous ratings and new ratings.

- Fuzzy models: in these models (e.g., [134,135,155]) trust and reputation is represented as fuzzy concepts that measures the degree to which an entity is trustworthy (reputable).

2.6 Summary

In this chapter, we presented fundamental concepts related to the subject of this thesis. We started by presenting web services and some of its underlying technologies with the description of the issues related to web services composition. we have seen that service composition can be achieved through service orchestration and service choreography. Service orchestration is where a central element controls all aspects of the implemented process. In fact, services orchestration is the most adopted technique in the industry using PBMN and WS-PBEL languages. We have seen that composition go through four phases during its life-cycle: (1) Definition of requirement and abstract process, (2) selection of services to concertize the defined abstract process, (3) deployment of the solution by binding services together, and (4) executing the services orchestration.

In the second section of this chapter, we described some information retrieval techniques used in the literature for measuring the similarity between documents. These techniques (e.g., TF-IDF, LSI) and similarity metrics (e.g., Jaccard, Euclidean distance, Cosine similarity) have shown their efficiency in text analysis and mining application. We have presented them because they are a key-elements

[7] http://www.ebay.com/
[8] http://www.amazon.com/
[9] http://www.epinions.com/

in the similarity measurement approach that we will describe in depth in Chapter 5.

Besides, we presented the Formal Concept Analysis method that is used for classifying objects and visualizing them into lattices. We illustrated the use of FCA by giving a real example on how to classify programming language. We limited the FCA presentation to the initial formal definitions, ignoring other FCA aspects such as queries formulation and navigation algorithms. We gave references for more detailed readings. FCA is one of the used techniques for the classification and the identification of web service substitutes, as we will describe in Chapter 6.

In the last section, we have seen that trust and reputation are crucial concepts in building strong interactions between entities. Trust and reputation management system have been proposed in the literature to evaluate trust and reputation of elements (e.g., services, products, agents) by aggregating collected users feedbacks. In the context of web service, reputation is an indicator about how users are satisfied by functionality, reliability and QoS parameters of consumed web service. Thus, reputation could be used as criterion in selecting reliable web services. Therefore, we propose in Chapter 7 a reputation management framework that evaluate reputation values of web services and their providers.

In the next Chapter, we present a literature review about methods and approaches related to subjects of this thesis.

Chapter 3

Literature Review

Contents

3.1 Overview

In this chapter, we discuss the most significant and recent advances in the areas of web service composition that are central to the subject of this thesis. The first

section (Section 3.2) provides a survey about web service composition methods. The second section (Section 3.3) presents an overview about reliable methods for web service composition. The third section (Section 3.4) examines some fault recovery methods for web service composition. The fourth section (Section 3.5) lists the works conducted for measuring similarity between web services. The fifth section (Section 3.6) reports some lattice-based web service classification methods). The sixth section (Section 3.7) summarizes the methods that cover the management of web service trustworthiness and reputation.

Each of these sections, contains a discussion subsection of similarities and differences between related works and our contributions.

3.2 Web service composition methods

3.2.1 Methods

Web service Composition has gained a significant attention from researcher over the past decade. Many methods (e.g., [5,20,28,31,37,46,58,66,82,85,91,104,106, 110, 114, 119, 123, 141, 144, 191]) have been proposed to cope with the different composition challenges. Several surveys over-viewing these methods have been published (e.g., [51,56,111,136,146,153,158,162,202]). In the following we report some of these notable methods:

Casati *et al.* proposed a system, called e-Flow, that supports the specification, enactment, and management of Composite services, modeled as processes that are enacted by a service process engine [28]. Composite services are modeled by a graph that defines the execution sequences among nodes in the process. The graph defines the flow of service invocation, and it may include services, decision and event nodes: services nodes represent the innovation of basic or composite service; decision nodes specify alternatives and rules controlling the execution flow, while event nodes enable services process to send and receive several types of events. Arcs in the graph may be labeled with transition predicates defined over process data, meaning that as a node is completed, nodes are connected to outgoing arcs are executed only if the corresponding transition predicate evaluates to true. To cope with dynamic environments, e-Flow provides a number of features that support adaptive service provisioning including dynamic service selection, dynamic conversation selection, and generic nodes. Generic service nodes are introduced to support personalized service composition. In eFlow, the definition of a service node contains a search recipe represented in a query language. When a service node is invoked, a search recipe is executed in order to select a specific service. E-Flow focus on optimizing service selection at a task

level. However, the system do not explicitly support any QoS model.

Sheng *et Al.* proposed SELF-SERV: a platform for dynamic and peer-to-peer provisioning of composite web services [144]. In SELF-SERV web services are declaratively composed, and resulting composite services are executed in peer-to-peer and dynamic environment. The platform distinguish three type of services: (1) elementary services, (2) composite services and (3) service communities. An elementary web service (1) is an individual web accessible application that does not explicitly rely on another web service. A composite service (2) is an aggregation of multiple web services (components) specified in statecharts, data conversion rules and set of selection policies. A service community (3) is essentially a container of alternative services. It provides description of desired services without referring to any actual provider. At runtime, when a community receives a request for executing an operation, it dynamically selects one of its current members and it delegates the request to it. In this platform, coordinators control and monitor the execution of associated services. Based on routing table, the control is passed from a coordinator to the next coordinator when the associated service finishes its execution. In addition, SELF-SERV includes a quality of service model that facilitate the specification of non-functional properties of web services [194].

Keidl and Kemper present a context framework that facilitates the development and deployment of context-aware adaptable Web services [85]. The framework consists of two main parts: a distributed infrastructure, which transmits context between clients and Web services and manages the context processing, and the context types, which are the supported types of context information and which are extensible at any time. This framework is implemented in an open and distributed web service platform called ServiceGlobe [86, 87]. This platform proposes two approaches: dynamic service selection and dispatcher services. Dynamic service selection allows selecting Web service instances during runtime by means of semantic classifications. The selection can be influenced by specifying different constraints. The dispatcher service addresses load balancing and high availability of services. The dispatcher service implements an automatic replication mechanism which allows to install new services on idle hosts on behalf of the dispatcher. Moreover, the dispatcher forwards requests to different instances and hence it reduces the risk of the unavailability of the service and accelerates request processing because of load sharing.

Medjahid et Bouguettaya proposed a composability model for semantic web services [106]. In this model, the composability is checked through a set of rules organized into four levels: syntactic, static semantic, dynamique semantic

and qualitative level. Each rule compares a specific pair of attributes of interacting web services. The notion of composability degree and s-composability are introduced to cater for partial and total composability. The model is part of the WebDG project which enables a semi-automated composition of services. WebDG presents an ontology-based framework for organizing and describing semantic web services. The concept of service community is also introduced in WebDG, which is defined as an instance of an ontology called community ontology. The verification of composability between semantic web services id conducted using the composability model. Moreover, the project defines a set of algorithms that generate detailed descriptions of composite services from high-level specifications of composition demands. These high-level specifications are written using a language called Composite Service Specification Language (CSSL). In addition, A Quality of Composition (QoC) model is also introduced to assess the quality of the generated composite services, which allows the selection of an optimal composite plan from possible existing composition plans.

Orrions *et al.* proposed a Business Collaboration Development Framework (BCDF) [123]. The framework adopts a rule driven mechanism which assists designer in the flexible development and adaptive management of business collaborations. This development process is organized in four phases including: (1) abstract definition, (2) scheduling, (3) construction, and (4) execution. The framework is composed of two principal components: Service Composition Repository which facilitates the management of composition elements and rules that are employed for service composition. And, Service Composition Manager which interacts with the first component to assist developing and executing composite services.

Lécué *et al.* propose an integrated approach to service composition [91]. The approach is designed within the context of SOA4All project. The approach consists of: (1) an automatic template process generator that is able to generate abstract process templates and their hierarchy from past executions. (2) A novel and scalable approach to AI parametric-design techniques using a multi agent approach to configure and adapt services processes, relying on the latter set of abstract process templates; (3) an optimization process that maximizes the overall quality of final compositions. In fact, SOA4all is large scale integrating project funded by the European framework Program. In SOA4All, available Web services, including RESTful services and SOAP-based services, are annotated with semantic information, intending to implement automated service discovery, mediation and composition. Contextual information, including local environmental constraints, organizational policies and personal preferences are considered in services composition. SOA4All uses a mashup-like way to assist end-users to

compose Web services.

Haddad *et al.* presented a transactional and QoS-aware selection algorithm for web services composition [58]. This algorithm guarantees that each selected component WS of a composite one is locally the best QoS WS among all the WS fulfilling the global transaction requirement. Transactional requirement is expressed by a risk notion that denotes if the results could be compensated or not. Quality requirement is described as a set of weights over QoS criteria. The authors also present and formally analyze the service selection algorithm based on the workflow patterns and the transactional properties of the component services.

Ardagna *et al.* presented a framework for specifying and managing flexible and adaptive composite services [5]. The framework provides a set of tools to support specification of all required information for runtime adaption of the composite services. The service selection method proposed in this work is based on mixed-integer linear programming model, which uses negotiation technique to bargain QoS parameters with service providers when an end user has severe QoS constraints and thus available solutions cannot be found. The framework provides mediation support if the selected service's interface differs from the interface that the corresponding task definition requires. The framework also provides supervision rules to monitor service execution and trigger corrective actions if needed.

Michlmayr *et al.* propose an execution environment for web service composition [110]. This runtime environment is represented as an application server containing (1) Query Engine, (2) a Notification Engine, (3) a Publishing/Metadata Service, (4) a Management Service and (5) a Composition Engine. The environment support both Soap-based and RESTful web services. In addition, it contains a QoS monitor that watches executed services performances, accuracy and availability. Moreover, Services versioning, dynamic QoS-based service selection, binding and invocation, as well as service mediation are all supported by the proposed solution.

Fujii *et al.* presented a semantics-based context-aware dynamic service composition framework that composes an application through combining distributed components based on the semantics of components and user queries formulated in natural language [66]. The framework is composed of (1) Component Service Model with Semantics (CoSMoS), (2) Component Runtime Environment (CoRE), and (3) Semantic Graph based Service Composition (SeGSeC). CoSMoS is a component model supporting function, semantic, and context representation of services. CoRE is a middleware that supports CoSMoS on different

distributed computing technologies. SeGSeC provides a mechanism to construct a composite service by synthesizing its process based on the semantics and contexts of component services. SeGSeC implements context-aware services composition based on specified rules or user preferences obtained through learning. The proposed framework acquires user preferences from user-specified rules and also via learning. The proposed framework also adapts to dynamic environments by autonomously composing a new application upon detecting context change.

Colombo *et al.* proposed SCENE: a service composition execution environment supporting dynamic changes disciplined through rules [37]. SCENE is a part of a European project called SeCSE (Service Centric Systems Engineering) aiming at providing methods, tools, and platform to support service oriented engineering. In this work, composition of services are described using SCENE language, through which composition is described in terms of two distinct parts: (1) a process part, described using WS-BPEL that defines the main business logic of the composition, and (2) a declarative part, described using ECA (Event Condition Action) rules. Rules are used to associate a BPEL workflow with the declaration of the policy to be used during (re)configuration. Rules can either be defined at design time or later before the execution of the composition. Various sets of rules can coexist and be activated depending on the preferences of the system users. The implementation of SCENE integrates an off-the-shelf BPEL executor engine called PXE, and a rule engine called Drools. A monitoring system is also integrated to provide SCENE with the required monitoring feedbacks.

Moser *et al.* presented a system, called VieDAME, which allows monitoring service QoS attributes and dynamic adaptation of BPEL processes [114]. VieDAME allows the replacement of existing partner services based on various (pluggable) replacement strategies, such as availability and response time. The chosen replacement services can be syntactically or semantically equivalent to the BPEL interface. Services can be automatically replaced at runtime without any downtime of the overall system. The dynamic adaptation mechanism is implemented with an aspect-oriented approach by intercepting messages between the composite service and participant services. The system is built from a component Core (VieDAME), and engine adapters components (VieDAME engine adapters). The core is responsible for monitoring, service selection and message transformation. The engine adapters offer the aspect-oriented interfaces to integrate different BPEL engines.

3.2.2 Discussion

In section 3.2, we presented a set of web service composition frameworks, methods and execution environments. Although these methods have contributed in solving domain-related problems and challenges, some issues are still open for research including:

- Reliable web service composition: Web services are normally distributed and autonomously provided by different organizations. Therefor, providing reliable and dependable services composition still remains a significant challenge [146, 199], especially for critical application; such as health-care, stock trading, air traffic control applications.

- Adaptable and autonomous web services composition: nowadays, the development and execution of composite web services is more open, dynamic and ever changing. Thus, there is a need for more adaptable and flexible approach for services composition [146]. Autonomous services composition is a promising research effort to increase the adaptability of services composition, which includes several fundamental properties: self-configuring, self-optimizing, self-healing, and self-adapting [128, 145].

- QoS awareness: Often, Composite service consumers are not interested only on the functionality of the service, they are interested also in the quality and the behavior of the service. Hence, composite services should be aware of their QoS aspects and those of the different elementary services involved [126].

- Risk awareness: Since the QoS of the composite service can be affected by the problems and QoS of its components, The service composition has to be aware of risks raised in such cases. Thus, there must be a mechanism or an action to mitigate it, for example, negotiating Quality of Service with partners or invoking other services [48].

- Security: Web services enable users to interact with internal applications and databases through the Internet, which represents a security risk. Services should be concerned about security aspects including authentication, authorization, confidentiality, and integrity to protect sensitive information [48].

The architecture that we propose contributes in assisting stakeholders to ensure and maintain reliable web services compositions by facilitating the selection of reliable services and reliables substitutes during different composition lifecycle's phase (See Chapter 6). Thus, in case of risks engendered by failure or

reliability degradation in the orchestration, the system can reconfigure itself, or the stakeholder upgrade the flow and executes one of the recovery plans built using the identified substitutes.

3.3 Reliable web service compositions methods

3.3.1 Methods

Many approaches and frameworks studied reliability for web services composition. Immonen *et al.* [73] presents a comprehensive survey about the following methods.

Cortellessa and Grassi proposed an approach for reliability modeling and analysis of service-oriented architectures [41]. They propose a methodology for reliability modeling and analysis in SOA. The reliability model is a prediction model presented as a probabilistic flow graph which contains statistical information needed to support reliability prediction. The flow graph includes requests from services to author services and information about the internal reliability associated to each service in the flow graph. Transition between nodes in the graph follow the Markov property. However, they extended it with other types of control flows that allow more than one external service request to be specified in each node.

The algorithm used for model evaluation takes in input the client perspective on reliability, where service reliability could be expressed by multiplying the probability (calculated using Markov process) of each node in the path to reach the end state of the flow graph.

In addition, the authors present an architecture that implements this methodology in SOA environments. It is important to note, that composite service provider has to publish information concerning the service internal structure, that is how external service are exploited, how service are glued together, and how frequency they are invoked. The methodology defines three service selection policies based on the published information. The selection procedure is supported by this methodology by comparing reliability of the concrete services and selecting the better reliable services.

Tari et al. Tari et al. proposed a framework for context-aware dynamic service composition in ubiquitous environment [161]. The authors aim to enable effortless integration of smart objects in a ubiquitous space. The approach that they propose provides a design architecture that includes a set of planning

algorithms and a mechanism for monitoring dynamic service composition. The authors distinguish between the concept of abstract and concrete service. The approach defines service composition architecture by mean of three plans: (1) abstract template, (2) optimal plans, and (3) a concrete execution plan. The template plan is constructed using rule-based techniques. This plan contains all possible abstract services that could compose the service. The optimal plan is created by selecting the best abstract services candidates according to their reputation and the complementarity of the required parameters. The execution plan is created by selecting concrete services based on their quality of user experience (QoE) value, which is weighted according to user preferences, user's context and the environment context. This plan is monitored. In this approach, adaptations enables substituting concrete services by other service, or even updating the optimal plan as whole. In this work, the authors introduce a QoS-based learning mechanism, which rewards a concrete service after execution, and calculates its new quality parameters and estimates the new reputation of its abstract service accordingly.

Cardellini et al. The authors propose a run-time adaptation method of service-oriented architectures [27]. This approach provides a methodology for runtime adaptation of services system in order to meet its QoS requirements in its operating environment. Two-level grammar is used to model the class of SOA systems managed by MOSES framework (MOdel-based SElf-adaptation of SOA systems): The first level specifies the structure of the composite service to be used. The second level defines the production rules for each abstract service. MOSES provides the idea of binding each abstract service to a set of functionally-equivalent concrete services. Thus, it requires as an input a set of candidate concrete services, and the description of the composite service written with a workflow orchestration language (WS-BPEL for instance). Once the description is verified, the behavioral model of the composite service is generated. The monitoring activity should detects any relevant changes occur in the operating environment. Therefore, the model is dynamically used to calculate possible arrangement using the available concrete service.

The authors propose to calculate QoS parameters of composite service by recursive rules using QoS parameter of the concrete services, in the way they are orchestrated. a behavioral model is used to build a template of an optimization problem, in which parameters are derived from the SLAs negotiated with the composite service clients and providers, and from the monitoring activity. The adaptation policy focuses on selecting the best possible implementation of the composite service in a given scenario optimized within a given environment. The framework can be applied to any composite service whose orchestration

pattern matches the first level of the grammar. A prototype of the MOSES implementation is provided by the authors.

Wang *et al.* propose a hierarchical reliability model for service based software system [175]. The approach focuses on analyzing reliability of (1) data, (2) service, (3) service pool and (4) composition. In this framework, the composition of services is specified as a workflow of processes. The reliability model is constructed of atomic, simple and composite processes that are connected by control constructs and set of transition rules. There are two points of view for modeling the overall system reliability: (1) The static model could be used before service binding and execution for early stage quality prediction. This model is generated by transforming the service process model into discrete time Markov chain (DTMC) model. (2) The dynamic DTMC model is dynamically constructed by run-time monitoring of the service execution paths. Execution monitoring detects occurred changes in system compositions, configurations and operational profiles. Thus, it adjusts the reliability model accordingly. The authors use what called service pool mechanism to provide runtime service redundancy, and to maintain local indexes of the available backup services. The authors implement a prototype that can automatically establish and adjust reliability models. The approach that they propose is generic and not specific for any particular domain, but it requires that services are described with OWL-S language.

Chawla et al. propose a real-time reliability model for ontology-based dynamic Web service composition [32]. The authors propose a real-time reliability model, in which reliability of a service is expressed using an OWL-S profile associated to each service. The OWL-S profile contains two parameters one for the desired reliability and the second for marginal reliability. The authors describe services as processes using a Process Model Template (PMT) into which suitable services are searched. The PMT is defined as a dynamic process model made of structural components. The model is then instantiated into instantiated process model (IPM) by binding components of PMT into concrete atomic or composite services. IPM extends the PMT with a set of placeholders for the details how a simple component can be bound to a selected Web service.

In this model, the atomic service reliability consists of the reliability of the service and the reliability of the machine where the service is deployed. The atomic service real-time reliability is calculated using the failure intensity and execution time. The hardware reliability is estimated using the shape and scale parameters. Parameters for calculating reliability are stored in the OWL-S profile associated to the service. The reliability of a composite service depends on its structure, the degree of independence between service components and the

availability of its constitutive Web services. Reliability model for each structural component (sequence, parallel, choice and loop) is defined. The approach supports maintaining reliability at runtime by monitoring the service reliabilities in real time. However, the reconfiguration of t he composite service requires human interference. A prototype of service monitoring tool is provided.

Hwang et al. they propose a method for dynamic Web service selection for reliable Web service composition [72]. The method is based on aggregated reliability (AR) metrics which are used to measure the reliability of each configuration in a WS composition. The method is also based on two dynamic strategies employed to computed ARs for dynamically selecting atomic Web services for web service composition. The service composition is described using Markov chains with added states, success, failure and transition probability. The aggregated reliability of each configuration is defined recursively from the probability that the services are successfully executed in the current configuration. Two Selection strategy are defined in this method: (1)AR-based selection strategy, and (2) composability and AR (CAR)-based selection strategy. In AR-based selection strategy, an atomic Web service is selected for each incoming operation of the composite web service to achieve a maximum reliability. In composability and AR (CAR)-based selection strategy, the ARs as well as the of configurations in selecting atomic Web service are considered. The method uses an iterative approach to compute the vector of aggregated reliabilities considering the different possible mappings in a Web Service composition to finally choose which sequence of service delegation to use. This method could be implemented using current web service standards and technologies. However, due to the nature of invocation orders of operations, it is required to use one of the business process composition language. The authors have developed a prototype using WS-BPEL.

Zeng et al. proposed an approach providing QoS-aware middle-ware that supports quality driven Web service composition [195]. The approach proposes a service quality model to evaluate the quality of atomic and composite web services. In addition, it offers two alternative service selection schemes for executing composite services. The quality model defines the QoS criteria for both atomic services and composite services. In this approach, the user assists the selection process by assigning wights to each selection criteria and assigning a set of user-defined constraints.

In the local optimization scheme, the optimal service selection is performed for each task in the service composition, without considering the global QoS. QoS informations of each candidate service are collected. Then, the system compute a quality vector for each service candidate. The service is selected basing on the

quality vectors applying a multiple criteria decision making technique.

In the global planning scheme, QoS constraints and preferences associated to a composite service as a whole are considered for service selection. Every possible execution plan associated with a given execution path is generated. Then, The selection of the appropriate execution is made by relying to the Multiple Criteria Decision Making technique. To select the optimal service or the optimal execution plan, the simple additive weighting technique is used in both approaches.

Moreover, the approach includes an adaptive execution engine, which reacts to changes occurring during the execution of composite services (e.g., a service become unavailable or its predicted QoS are changed) by re-planning the execution. The approach has been implemented as a platform providing tools for defining service ontologies, specifying composite services with state charts, and assigning services to handle tasks in service composition.

Ma and Chen proposed a reliability evaluation framework for composite Web services [98]. The approach proposes a service reliability model for both elementary Web services and composite services. In addition, the authors proposed a feedback-based composite service approach. The reliability of elementary services is calculated using time-dependent Markov model, with failure tensity, failure locating and fixing time.

In this approach, the composite service structure is describes as graph of nodes and relationships between nodes. Markov chain is used to evaluate the backup services to ensures node's reliability.

Finally, the aggregated composite service reliability is presented as reliability of nodes and the operation relationships of subset of node set. The feedback-based framework employs feedback mechanism to collect QoS from clients that consumed the service. The automatic collected QOS are stored in UDDI registry. The model uses this data for service reliability evaluation each time a change happen in the service composition during its execution.

3.3.2 Discussion

In the previous section, we over-viewed a set of methods that supports reliable web services composition. These methods have addressed reliability during design time. They proposed multiple technique for analyzing and verifying reliability. However, none of these methods considered the means to verify reliability during run-time and maintenance phases.

In addition, most of these methods do not include reputation management of services and providers as an indicator factor about the service reliability. Reputation indicates how services are experienced by users, and what is the level of satisfaction of these users about service offered functionalities and qualities. Thus, it is an important factor to be included for ensuing reliable web service composition.

Moreover, our proposition is flexible when it identifies reliable substitutes and allowing the built of recovery plans, which could be updated dynamically during runtime when reliability informations are available or relevant new services have discovered.

3.4 Fault recovery in web service composition

3.4.1 Methods

Different kinds of faults may occur during service composition execution, and many strategies were proposed to repair the failed services.

In [65], authors present an approach for fault management in Web application. The contribution is a self-healing system that holds all possible faults and their repair actions in a special registry. Authors present reference architecture for faults treatments and a set of strategies for recovery. Moreover, a classification of faults have been studies and schematized. The core of the approach is based on searching substitutive services for repairing compositions.

In [52], authors propose to transform a BPEL process into a fault tolerance process using a fault tolerance patterns. The transformation is achieved by adding redundant behavior to the application.

Baresi et al. in [16] present a supervision framework and a solution for self-healing BPEL process based on Dynamo [15]. The framework lies on the use of Aspect oriented programming techniques, separation of concerns principals and rule engine (JBoss rules technology) in order to allow recovery of faults in service composition.

WS-Diamond [40] is a project for self-healing web services. It is based on a platform for observing symptoms in complex composed applications. It aims to diagnosis occurring faults, and for selection and execution of repair plans.

In [148] and [147] propose a framework for performing runtime monitoring of web services applications against behavioral correctness properties described as finite-state automata. The set of verified properties specify forbidden and desired

interaction between services. The execution traces of web services applications described in BPEL are checked for conformance at runtime. The framework proposes different adaptation strategies in case of violation of properties.

In [45], authors introduce Qos-driven self-healig method for reliable web service composition. The method predicts Qos and performances during composition. It backups alternate web services during the selection step. Then, in case of failure, it reselect from these backups based on Qos and performance predictions. Moreover, authors use a Semi-Markov process to predict the data transmission speed over the network where services are executing.

3.4.2 Discussion

In the previous section, we listed a set of methods that addressed the problem of fault recovery in web services composition. Similarly to these methods, we have proposed an approach that allows selection of substitutes based on similarity measurement between a web service and a set of potential candidates. These substitutes that could be simple or complex serve as backups for different partners in services orchestration. Hence, designers can statically construct different execution scenarios based on the identified substitutes, as well as these substitutes are dynamically updated during runtime. These scenarios are recovery plans used in case the orchestration fails.

3.5 Similarity measurement for service discovery and selection

3.5.1 Methods

The similarity evaluation between web services has been studied by many researchers for service discovery and selection. A survey with a comparative analysis between proposed approaches is found in [88] and [43]. Most of these works uses Information retrieval (IR) techniques to increase web service discovery precision without involving any additional level of semantic mark-up [68].

In [47], authors present their tool ARTEMIS which calculates a set of similarity coefficients to evaluate web service compatibility. The tool clusters similar services based on the obtained similarity coefficients.

The paper of [53], presents a search engine called "Woogle". Based on similarity search, Woogle returns similar Web services for a given query. The search engine combines multiple techniques to evaluate similarity between the services

and their operations. These techniques focus on operation parameters as well as operations and services description. The authors introduced a clustering algorithm for grouping description terms in a set of concepts. After that, similarity between concepts is measured using a simple information retrieval metric; the TF/IDF metric.

The similarity evaluation in [88] is implemented through the combination if lexical and structural matching. Likewise, in [133], the paper proposes a method of Web service retrieval called URBE (Uddi Registry By Example). The retrieval is based on the evaluation of similarity between Web service interfaces. The algorithm used in URBE combines the analysis of Web services structure and the terms used inside it.

Authors in [1] studied the similarity measurement between behavioral interfaces of web services by simulation. Both, structure and behavioral aspects of service are considered in this study. Structure aspects are presented by service operation, messages, and their schema within the interface description document. And, the behavioral aspect, which is presented by finite state machine, is defined by control flow and interdependencies between operations.

The approach presented in [42] proposes to discover the most relevant web service to a given query. The approach is based on the representation of a web service description and queries within classic space vectors. Then, it matches between the vectors that represent services and the vector which represents the query using the Cosine metric. It returns the nearest service to the given query.

In addition to the previous works, an approach for measuring the compatibility degree of services' protocols is proposed in [124]. The approach relies on formal comparison that is based on generic-flooding based-technique. Authors provide a formal model for describing web service interface with interaction protocols.

Another approach is proposed in [68] for service selection. The approach comprises an assessment process for service interface compatibility. The assessment process is based on structural scheme for service matching. The scheme is divided into two main parts: automatic strong matching and semi-automatic potential matching. The former involves similarity cases directly recognized from Java interfaces of candidate services. The latter involves cases that could be solved through a semi-automatic assistance. The whole information package gathered from this process provides an important insight about candidate services and their required adaptations for integration.

3.5.2 Discussion

The similarity evaluation between web services has been studied by many researchers for service discovery and selection, as we have seen in section 3.5. We compared these methods and ours using the following metrics :

- **Syntactic**: whether the work considers syntactic matchmaking or not.
- **Semantic**:whether the work considers semantic matchmaking or not.
- **XML-Schema structure**: whether the work analyses XML-schema or convert them.
- **Similarity Flooding**: whether the work uses a similarity flooding technique.
- **Similarity relationships**: whether solution provides a detailed similarity measurement between services elements.
- **Composability relationships**: whether the work identifies composability relationships between operations or services.
- **Clustering**: whether the solution clusters services based on similarity.
- **Formal method**: whether the work uses formal methods to describe services before matchmaking.
- **Behavioral aspects**: whether the work take in consideration behavioral aspects of web services during similarity measurement.

Table 3.1 summarizes the similarity assessment approaches based on the criteria fixed above, with a comparison with our similarity measurement approach.

Our approach uses different structure and semantic similarity metrics to conduct a similarity assessment between different WSDL parts of the compared web services (operation, messages, parameters, type, etc.) It uses also a schema matching technique to evaluate the similarity between input and output messages. The similarity function is parameterized by a set of weights to allow users determine which parts of a WSDL document have more impact on the similarity score. We adopt this approach in the current work for the similarity assessment between web services. Nevertheless, the identification process of service substitutes that we presented on Chapter 6, and which uses the similarity approach, is a generic, and it is not limited by the use of a specific similarity measurement approach. So, any other measurement approach, such as [68,88,133], could be incorporated during the similarity matrix construction between service candidates (See Chapter 6).

Table 3.1: Similarity methods comparison

	Syntactic	Semantic	XML schema Structure	Similarity flooding	Tree tagging	Similarity relationships	Composability relationships	Clustering	Formal method	Behavioral aspects
De Antonellis et al. [47]	✓	✓	✓			✓		✓		
Dong et al. [53]	✓	✓	✓			✓		✓		
Kokash et al. [88]	✓	✓	✓			✓		✓		
Plebani et Perenici [133]	✓	✓	✓			✓		✓		
Aît Bachir et al. [1]	✓		✓			✓				✓
Crasso et al. [42]	✓				✓	✓				
Ouderni et al. [124]	✓			✓		✓		✓	✓	
Garriga et al. [68]	✓		✓			✓		✓		
Our Similarity approach	✓	✓	✓	✓	✓	✓	✓	✓		

3.6 Lattice-based web service classification

3.6.1 Methods

Many works have addressed the classification of web services using concept lattices. In their paper [130], authors present a formal definition of web service classification and retrieval using formal concept analysis. The essential of the approach is to build lattices using formal concepts where object are web services and attributes represent the operation of these services. Then, they retrieve similar services using algorithms that navigate the elaborated lattices.

In order to understand relationships between web services, and among operations of complex services [7] propose an approach based on Formal Concept Analysis. The approach analyzes service interfaces and documentations to construct lattices. Generated lattices allow analysts to clusters similar services, highlight hierarchical relationships, and visualize, in general, similarities and differences between the analyzed services.

Restructuring of services registry at runtime using Formal Concept Analysis is studied in [35]. The purpose behind this approach, as authors claim, is to speed

up the selection process of web services, and to improve decision making through the building of concept lattice. The services registry is viewed as formal context where services are objects and service types, functional, and non-functional characteristics (security) are the context properties.

The work in [62] describes an approach for retrieving semantic web services, taking in account user requirements and preferences. The approach exploits the fuzzy formal concept analysis for modeling concepts and relationships extracted from web service resources. User formulates its query as conceptual terms, the system formulates the query and through a conceptual based mechanism it returns the list of semantic web services that matches the introduced query.

Authors in [54] introduce a requirement-centric approach that allows modeling user requirements, discovering and selecting web services. Authors use formal concept analysis only for selecting automatically relevant high QoS services.

In addition, authors in [11] elaborates a tool named WSPAB. The tool uses formal concept analysis to allow automatic discovery, classification, and selection of web services. The tool build formal concepts where objects are web services and attributes are operation signatures. Then, it generates correspondent lattices that classify studied web services. Same authors propose another approach to classify web services by keywords elicited from their wsdl documents [12]. The approach clusters similar services, so it is possible to identify relevant services and their substitutes.

Moreover, a Rational Concept Analysis approach is proposed to select composable web services driven by user requirements [9]. RCA is an extension of formal concept analysis. The approach is based on four principal steps including: Service Collecting, validity and compatibility filtering, Qos level calculation, and RCA classification. The resulting lattices group services that have common QoS and composition levels. User requirements are expressed as new services and are classified in the corresponding lattices.

3.6.2 Discussion

In section 3.6, we presented different methods that use FCA, and RCA for classifying and selecting web services. In this section, we made a comparison between these methods based on the following criteria:

- ✠ **Design phase**: whether the method is used at design phase, or not.
- ✠ **Runtime phase**: whether the method is used at runtime phase, or not.

- **Maintenance phase**: whether the method is used at maintenance phase, or not.
- **Global reliability and QoS analysis**: whether the method analyses the overall orchestration QoS and reliability, or not.
- **Elemantry reliability and QoS analysis**: whether the method analyses QoS and reliability value of each atomic service in the orchestration, or not.
- **Clustering**: whether the method clusters web services or not.
- **Composability relationships**: whether the method is able to detect composability relationships between web services or not.
- **Similarity relationships**: whether the method is able to detect similarity relationships between web services or not.
- **Simple Substitutes**: whether the method is able to detect simple service substitutes or not.
- **Complex Substitutes**: whether the method is able to detect complex service substitutes or not.
- **Semantic Web services**: whether the method supports Semantic web services or not.
- **Lattice-based**: whether it is a lattice-based web service classification method or not.

Table 3.2 summarizes the presented lattice-based web service classification methods, according to the criteria presented above.

As we can see, the propositions of Azmah *et al.* [10] are the nearest to our second contribution (Chapter 6). In the work of Azmeh et al., the user has to introduce a requirement document that specifies an abstract process with the needed functionality and the expected QoS in each service, as well as the composability between each pair of services. Then, the approach selects concrete web services for each element in the abstract process. The selected services are simple and composable with each other according to the composability modes described initially in the requirement document. Certainly, this approach selects elements for building compositions, but the selected services are all simple. Moreover, the solution is static (number of services in the process is fixed) and totally guided by the user. In the opposite, the solution that we propose is used at the maintenance phase. It is dynamic, i.e. the approach retrieves service substitutes with a variable number of services in the composition for the same specification.

Table 3.2: Lattice-based service classification methods comparison

	Life-Cycle			QoS		Hierarchical			substitutes			
	Design Phase	Run-time phase	Maintenance	Global reliability and QoS analysis	Elementary reliability and QoS analysis	Clustering	Composability relationships	Similarity relationships	Simple	Complex	Semantic Web Service	Lattice-Based
Peng et Chen [130]	✓							✓	✓			✓
Aversabo et al. [7]	✓					✓		✓				✓
Chollet et al. [35]	✓				✓				✓			✓
Fanza et Senator [62]	✓							✓	✓		✓	✓
Driss et al. [54]	✓				✓				✓			✓
Azmah et al., [9, 11, 12]	✓				✓	✓		✓	✓			✓
Our approach	✓		✓		✓	✓	✓	✓	✓	✓		✓

In addition, the approach does not need any abstract description to guide the identification process; the approach browses automatically all composition possibilities and finds appropriate substitutes for a failed service described by its WSDL document.

Technically, we have updated the selection and filtering phases by proposing new similarity-based algorithms. Then, we used a more complex similarity method to measure the similarity between web service candidates. We focus on the similarity between input and output messages rather than the similarity between operations like Azmeh et al. did in their paper. The study of similarity between service inputs and outputs can reveal both composition and similarity relationships, but the similarity between operations reveals only similarity relationships which leads to the discovery of simple substitutes only. We also performed additional tasks for the classification of services (e.g. Group extending and the steps applied in this task and those applied during the interpretation task). This refined process enabled us to construct lattices that cluster operations

and show their potential simple and complex substitutes.

3.7 Reputation management models

3.7.1 Methods

Reputation management has been extensively studied in different computer science areas including E-business (e.g. [64, 75, 95]), Multi-agent systems (e.g. [25, 76, 131]), Peer to peer (P2P) networks (e.g. [61, 198, 201]), grid systems (e.g. [38,172]), mobile and ad-hoc networking (e.g., [34,137,157]). In this section, we give an overview about some proposed reputation management approaches in the context of web services. Comprehensive literature reviews are available in [70, 80, 103, 122, 177].

Conner *et al.* introduced a reputation-based trust management framework that supports the synthesis of trust-related feedback from many services on the basis of previous interactions with clients [39]. The core of this framework is a centralized reputation manager that allows services to compute their own customized reputation scoring functions over the records of the collected feedback. The framework provides a trust-evaluation caching mechanism using the Bloom filter [23] and Bloom histograms in order to reduce the communication overhead in the system. The main advantage of this approach is that it supports multiple reputation measurement models which is adequate to multiple web service environments. However, the approach does not offer a strong mechanism to find malicious feedback ratings and to distinguish malicious users among other users. Moreover, the approach suffers from getting accurate reputation values due to the calculation of trustworthiness when good feedback users become bad or bad users become good [176].

Bianucilli *et al.* proposed a generic and customizable reputation framework to automatically and transparently monitor the execution of composite web services [21], tacking into account both functional and non-functional properties. This framework is built upon a client server architecture. The server side comprises three components; 1)an enhanced UDDI-based registry, 2)a reputation manager to compute service reputation ranks based on collected QoS values, and 3)a subscription manager to notify subscribed service consumers about occurring changes in service reputation. The client side comprises: 1) a monitor component that monitors the behavior of partner links in BPEL orchestrations by verifying some functional and quality assertions. 2) a reputation feeder which is responsible for the collection of service feedbacks and reports them to the reputation manager in the server side. 3) an event manager which provides functionalities

to subscribe to reputation-related events and to react to such notifications. The main component in this architecture is the reputation manager which has been designed to support different methods for computing service reputation through the installation of new reputation policies as plug-ins. However, we have to mention that the default reputation assessment formula do not include the user credibility nor the impact of the rating time on the assessment of reputation scores.

Mokarizadeh *et al.* presented a selection framework for service orchestrations based on the trustworthiness of services [113]. The approach measures the trustworthiness using the reputation scores retrieved from users' profiles. The profiles are extracted and inferred from a social network which accumulates users past experience with corresponding services. The approach applies a T-index technique [193] to compute the trust values between users. in addition, it employs a privacy inference model to protect sensitive information in social network, when analysis its content to extract the corresponding reputation values.

Malik and Bouguettaya proposed RateWeb [101], a decentralized reputation system for web service orchestrations. The proposed architecture is based on a peer to peer (P2P) service model where each peer (service) is a consumer and a provider of services. The solution is characterized by the absence of central reputation management entity. Thus, each peer in the system is responsible for collecting, updating and calculating the reputation of the other peers. Hence, each peer has its own view of the reputation of other services. RateWeb uses an ontology-based community model. This community is viewed as a directory of raters which allows consumers of services to get information about raters. When consumer services decide to get reputation information about a certain service, they directly ask the raters of these services. Moreover, the framework takes into account the presence of malicious raters that may exhibit oscillating honest and dishonest behavior. The temporal sensitivity is also considered in this work. Hence the system fades the values of old ratings, so the system can consider and give more weights to the newest service rates. One of the strengths of the approach is that it is based on the inclusion of the different factors mentioned above. However, the approach is not applicable as it is proposed for an open recommender service system that can hold thousands of services which cannot act as consumers and providers at the same time.

Limam and Boutaba [94] proposed a framework, derived from the expectancy dis-confirmation theory from market science, for reputation-aware service selection and rating. The key characteristic of the proposed framework is to

automate the selection and the rating of services. First, the authors have proposed a rating function that makes use of quality monitoring results and service cost to produce feedback. Second, they have introduced a reputation derivation model that aggregates all of the feedbacks into a reputation value. Finally, the authors have presented a selection function that derives a single selection metric out of the reputation of the service as provided by the reputation system and the offered quality and cost. However, it is still difficult to predict the accurate feedback ratings in real web services environments, especially with the existence of malicious and subjective feedback behavior.

Mekouar *et al.* proposed a trust and reputation management system for peer-to-peer systems [108], and for web service environments (TrsutWs) [107]. TrustWs permits a selection of web service based on the feedback gathered from past transactions. In this work, authors distinguish between feedbacks issued from satisfactory transactions (positive feedbacks) and feedbacks issued from unsatisfactory ones (negative feedbacks). Hence, the proposed reputation scheme assesses the reputation of web service as the ratio of the difference between positive and negative feedbacks, and the sum of all feedbacks. However, this work do not provide any mechanism to deal with malicious users and subjective ratings, which make the system vulnerable to the Sybil attack for instance.

Wang *et al.* propose a reputation measurement and malicious feedback rating prevention approach for web service recommendation systems [176]. The goal of this approach is to reduce the deviation of the reputation measurement of web services, and to improve the success ratio of the service recommendation. The approach passes through two phases before computing services' reputation scores; the malicious feedback rating detection phase and the feedback rating adjustment phase. In the first phase, authors apply the Commutative Sum Method (called CUSUM) to detect all the malicious collected feedback ratings. In the second phase, the authors deal with the computation of feedback similarity between different users using the Pearson correlation coefficient to adjust the feedback ratings. After the second phase, the system measures the reputation of services based on the former ratings, and stores the calculated reputation scores in the repositories to be used for service recommendation. Moreover, the authors propose a Bloom filter-based prevention scheme to identify the IP addresses with offending feedback ratings and filter them out. However, the temporal sensitivity of feedback ratings is not addressed in this approach. The new feedback ratings have to influence the most on the reputation assessment than the old ratings, because the performance of any service may change over time (from high to low and vise versa), and only new feedback ratings can indicate its actual performance.

3.7.2 Discussion

To compare the presented reputation management methods with our fourth contribution, we use the following metrics:

- **Time Sensitivity**: whether the method considers the time of ratings in the evaluation of reputation or not (i.e older ratings have less impact on reputation than new ratings.)
- **User Credibility**: whether the method includes the credibility of users (i.e., Honesty degree) in the calculation of service reputation.
- **Reputation bootstrapping**: whether the method provides a solution for assigning initial reputation value to new-comer services or not.
- **User rating history**: whether the system includes all the ratings provided from the same users in assessing the reputation of the same item.
- **Penalization or punishment of malicious users**: Whether the system punishes malicious users or it does not.
- **Feedback history**: whether the method includes all the feedback history for assessing the reputation of services or not.
- **Personal experience/profile**: whether the method uses personal profile and experience information on the assessment of reputation or not.
- **Cold Start problem**: Whether the method proposes a solution to the cold start problem or not.
- **White Washing problem**: Whether the method proposes a solution to the white washing problem or not.
- **Autmatic evaluation**: whether the method is automatic or it needs a manual intervention.

Table 3.3 summarizes the presented reputation management methods according to the criteria presented above.

Although existing works have proposed some efficient and robust reputation management models, most of them suffer from the following shortcomings:

- First, consideration of fair ratings: it is still difficult to ensure the purity of users' feedback ratings [176]. In some circumstances, feedback ratings issued from unidentified malicious users have a negative impact on the assessment of web service reputation if these ratings are not neglected. Thus, the reputation assessment model has to successfully guard unfair ratings.

Table 3.3: Reputation management methods comparison

	Time Sensitivity	User credibility	Reputation bootstrapping	User rating history	Penalization of specious users	Feedback history	Personal experience/ profile	Cold start problem	White washing problem	Automatic evaluation
Conner et al. [39]		✓		✓		✓	✓			✓
Bianucilli et al. [21]				✓		✓				✓
Mokarizadeh et al. [113]				✓		✓	✓			✓
Mailk et Bouguettaya [101]	✓	✓	✓	✓		✓	✓	✓		✓
Limam and Boutaba [94]		✓	✓	✓		✓		✓		✓
Mekouar et al. [108]	✓			✓		✓				✓
Wang et al. [176]	✓	✓		✓	✓	✓				✓
Our work	✓	✓	✓	✓	✓	✓		✓	✓	✓

- Second, consideration of user credibility: credibilities of raters have to be carefully evaluated [121]. In open systems, participant users are unknown. Some of them are honest and behave correctly during the feedback process. Other users act maliciously and send unfair feedback ratings. A third type of users have their subjective preferences which influence sometimes on their judgments. Hence, an accurate model that assesses the credibility of user, based on the probability that user's feedback ratings are conformed to the major user rating tendencies, have to be proposed.

- Third, recommendation dissemination: based on the survey conducted by Mármol *et al.*, the description of how to gather and store feedback information is a commonly neglected factor [103]. Many authors are considering that information is available, without explaining how to collect and store them.

- Fourth, bootstrapping the system: most of reputation management frameworks present the algorithm for the assessment of service reputation score, obviating the bootstrapping of the system [102]. The attribution of arbitrary initial reputation values to newcomer services may cause harmful damages and make the system vulnerable to different threats (e.g., Sybil

attack) [103].

- Finally, considering reputation of web service orchestration: most reputation management proposition lack the consideration of reputation assessment of web service orchestration. An approach on how to estimate the overall reputation of web service orchestration would enhance the selection of web services for composition during design time, and the recommendation of substitutes for orchestration maintenance in case of failure.

In this thesis, we propose a reputation management framework for web service reputation that addresses the previous challenges. The assessment model that we propose includes a penalization mechanism of suspicious users, to ensure an efficient reputation measurement from pure feedback ratings. The assessment model is built upon the distinction between positive and negative feedback ratings, with the inclusion of time sensitivity and user credibility factors, where the former is evaluated according to majority consensus. The bootstrapping mechanism that we propose allows a better introduction of newly published web services into the system, because efficient initial reputation values stabilize the performance of the whole system. Finally, the approach is a centralized solution which is appropriate for building web service recommendation systems.

3.8 Summary

In this chapter, we presented thesis related work. First, we started by providing an over-view about web service composition methods. We have seen that providing reliable and dependable service composition still considered as a significant challenge for the most of the proposed methods.

Then, we listed the methods and approaches that focus on reliability issues during web services orchestrations. We have seen that these approaches have different meanings for reliability, varying from a functional and available web service to web service with high QoS parameters. However, most of these methods do not include reputation as a reliability factor during the selection of web service.

After that, we presented methods and approaches proposed in the literature for web service Selection, Similarity measurement, and FCA classification. All these methods and approaches are close to our contributions that we will present in next chapters. We have compared between our contributions and these works based on different sets of criteria. We have discussed differences and reported goals.

Finally, the chapter summarized the literature of web service reputation management by listing the notable proposed frameworks. We discussed the differences between these works and our reputation management framework. We used a set of criteria on which the comparison was made.

In the following chapter, a description of the life cycle of reliable orchestration is presented in the first section. Then a generic architecture is proposed for assisting designers and developers to construct reliable web service orchestrations and to maintain them.

Part II

Contributions

Chapter 4

Framework for Reliable Web Services Orchestration

Contents

4.1 Overview

The classical life-cycle of Web services orchestration (presented in Section 2.2.5) does not cover reliability analysis and verification in its different phases. Therefore, we refine the life-cycle to take in consideration reliability requirement, verification and analysis.

In addition, we propose in Section 4.3 an architecture that assists WS orchestration stakeholders (Designers, developers, maintainers ...) to build reliable

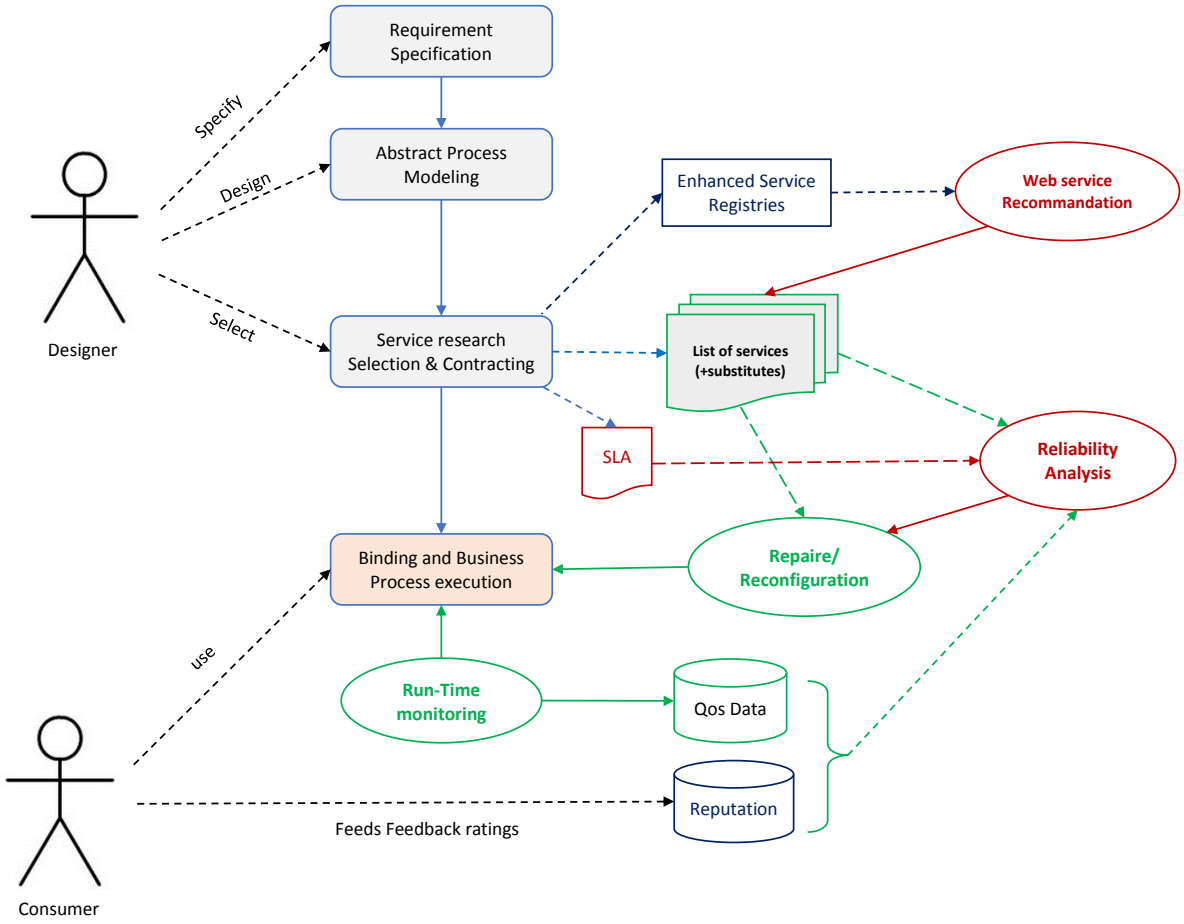

Figure 4.1: Reliable orchestration life-cycle

WS orchestration through the recommendation of reliable and reputable web services.

4.2 Reliable WS orchestrations life-cycle

In this chapter, we extend the classical web service composition life-cycle presented in section 2.2.5 with new phases to ensure reliable web service composition [1]. As depicted in Figure 4.1, to ensure a reliable web service orchestration, the later should passes through multiple phases explained in next subsections.

[1] In the rest of this manuscript, we will use the terms web services composition, composition, web service orchestration and orchestration interchangeably

4.2.1 Phase 1: Requirement specification

The design of service composition [2] starts by specifying functionalities and qualities that the to-be service (i.e., the orchestration) has to ensure. The designer has to write down a specification document that describes how and what does the to-be service do. And, it measures both qualitative and quantitative requirements ending with some design decisions.

Service Oriented Requirement Engineering (SORE) [57,170] methods should be applied during this phase for capturing eliciting and managing these requirements [14]. Though many SORE methods have been proposed (e.g., [169] [6] [182]), the adopted SORE method must allow the designer to express explicitly reliability requirements in form of metrics, which will be used to verify the fulfillment of reliability requirement during service runtime.

4.2.2 Phase 2: Abstract process modeling

Based on the specification document established in phase 1, orchestration designer establishes a service architecture that describes the to-be service in a form of abstract process (Or, service template) into which conceptual services can be searched in next phases and be replaced.

Reliability requirements specified as metrics in the previous phase have to be expressed in the service architecture in a formal way (as proposed in [89] for instance) as a design decision and a set of must-have qualities for some or all services in the abstract process.

4.2.3 Phase 3: Service search, selection and contracting

The orchestration designer performs a deep analysis on the service architecture in order to determine which parts from the abstract process could be implemented using ready-made (i.e., available) web services. Several web services offered by different providers could ensure the specified functionalities, and thus they can be a search-result candidates for the abstract process elements. The designer has to select among these candidates services that fulfill the non-functional requirements.

Generally, services repositories, from where the services are searched, contain usually static and ideally dynamic QoS data advertised by service providers or collected from service consumers. The designer conducts a reliability analysis

[2] For the rest of the document we use the words orchestration and composition interchangeably

to select high reliable services, because any service with low reliability decreases the reliability of the whole orchestration [73]. The designer has to be sure of the used quality data. Moreover, she/he must consider the trustworthiness of the services and their providers.

The selection of the principal services is achieved based on the reliability analysis. Though, the reliability and QoS are important data for selection, they are not enough because the weak reputation of a provider is a good reason not to select certain services. In contrast, the reputation of a service itself is a justification for selection since it indicates how well it has been experienced. Thus, the designer consider both reliability analysis data and reputation for principal service selection.

Eventually, The designer could establish contracts (e.g., SLA: Service Level Agreement) with service providers. Typically, such contract contains numerous service reliability metrics (i.e. qualities) which the provider has to guarantee.

Besides, the designer may select from the service candidate-list a set of backup services; each backup service is a substitute for the principal service in case of failure or the non-respect of the required reliability level. Moreover, she/he can make different recovery-plans based on the available service substitutes. So, in case of failure one of the recovery planes replaces the principal orchestration.

4.2.4 Phase 4: Binding and business process execution

In this phase, the service orchestration begins its execution. Selected services are bound according to the protocol specified in the service description files. The execution engine executes the interactions between services based on the business process (i.e orchestration) described with one of the service composition languages such as the WS-BPEL for example. When the orchestration is operational, consumers can start exploiting its capabilities.

4.2.5 Run-time monitoring

Since runtime failures of web services are inevitable, monitoring technique has to be applied to watch the behavior of web services during execution. The technique focuses on monitoring both functionalities (e.g., assertions) and QoS of services at runtime. Monitored data should be stored in special database for conducting farther reliability analysis. In addition to QoS information, the database holds consumer feedback ratings that indicates their experience with the executed services. The database has to hold the most recent monitored values,

and it should be available for anyone searching for dynamic Qos information about services.

4.2.6 Reliability analysis

To maintain the required reliability during orchestration runtime, a regular reliability analysis must take place through orchestration life-cycle. The reliability analysis uses monitored QoS information, assessed service similarity and provider reputation values to calculate the actual orchestration reliability. The SLA contract is monitored based on the reliability results. Thus, we can determine whether the SLA contract is preserved or its violation has occurred. In the later case, or when one of the orchestrated services became unreliable, a self/manual adaption and reconfiguration of the orchestration is needed.

4.2.7 Repair and reconfiguration

The result of the reliability analysis indicates which services are become unreliable or none functional. To ensure a self-adaptiveness and self-healing orchestrations, a special repair and reconfiguration unit has to take in charge the replacement of defected services. First, it checks the list of substitute candidates to inspect for relevant reliable services. Then, it reconfigures the business process based on the recovery-plans established at the service search, selection and contracting phase. Finally, the unit re-lunches (i.e., re-executes) the reconfigured orchestration.

4.2.8 Web service recommendation

During orchestration runtime, new services with high reliability values may be published or discovered by the special repositories. Hence, a continuous service recommendation with reliable services can help in preserving the specified reliability. Ideally, a special service recommendation unit should check services repositories and then updates the list of candidates with new reliable services. Moreover, it can update periodically candidate reliability and reputation information gathered from the monitoring and feedback databases.

4.3 Architecture for reliable WS orchestrations

In this section, we present an architecture for assisting stakeholders to design, execute and maintain reliable web services orchestrations. The architecture regroups a set of methods and techniques that allow the selections and recommendation of reliable web service and their substitutes. Both selection and recom-

mendation operations are ensured after the reliability analysis in which QoS and reputation metrics are evaluated.

The architecture permit to manage of consumer feedback-ratings, which allow evaluating reputation values of services and their providers. The reputation is an aggregation of subjective opinions about the overall quality (including reliability) of a used service.

Moreover, the architecture allows the notification of service providers by reputation values of their offered services. Thus, service provider can enhance services' performances accordingly.

Figure 4.2 shows the proposed architecture. As we can see, the architecture is composed from three main elements: (1) service provider, (2) service client (Orchestration stakeholder), and (3) Web service recommendation system. In the next subsections, we describe these components.

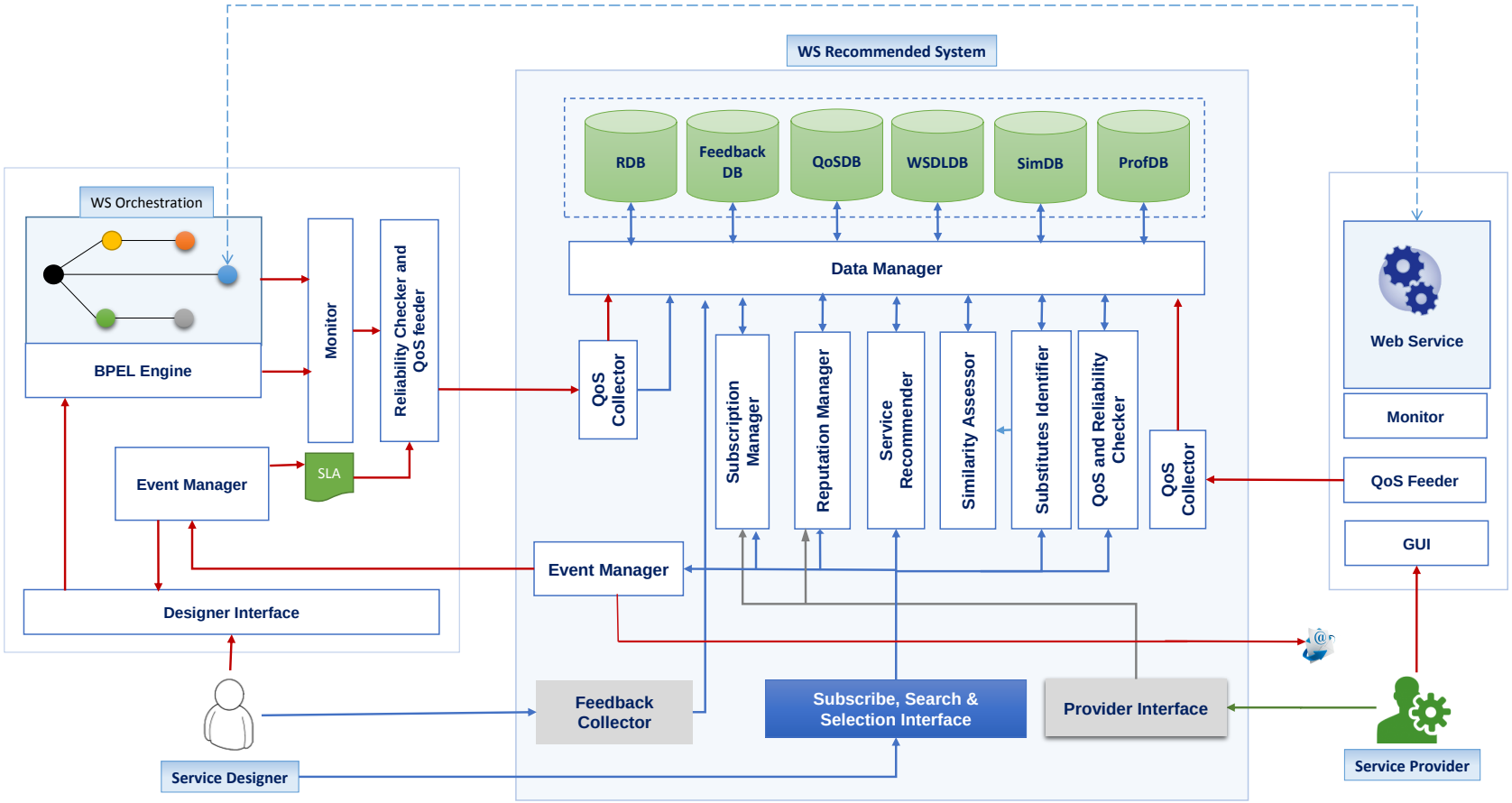

Figure 4.2: Architecture for reliable orchestration

4.3.1 Service provider

The service provider plays a central role in providing and maintaining high reliable service. In a failure-prone environment, the qualities of services variate according to many factors (e.g., deployment factors, network factors). In addition, the provider may need to follow reputation values (feedback)of its services, especially if these services are dedicated to a large audience. Therefore, service provider have to keep its service competitive by providing not only reliable functionalities but also by providing the better QoS possible.

As it is shown in Figure 4.2, the provider has a lightweight application that monitors deployed services. Ideally, monitoring results, which could be a vector of QoS values, have to be sent automatically, by a QoS feeder component, to the WS recommendation System. This should be done automatically once the provider subscribes in the WS recommendation system and publishes its service. The WS recommendation system permits to the provider to see how its service is perceived by consumers, and what reputation value it merits. Moreover, the system notifies providers when a violation occurred in the level of reliability of elementary services.

We summarize the tasks of the provider as follows:

- Subscribes to the system: The provider should subscribe to the WS recommendation system. It may be identified by its domain, IP address, or a identifier.
- Publishes a new Service: The provider may publish new services. He/She has to enter the name of the services, a set of keywords that describe the service, The URL of the service, she/he uploads its WSDL file, and provides the initial QoS of the service.
- Monitors its Services: Using the monitoring component, the provider have to keep watching the service that he provides.
- Updates its service quality report. The QoS feeder component, after configuration, could send automatically QoS reports to the WS recommendation system which stores these values.
- Checks statistics (global and for each service): The provider is able to follow its statistics by consulting its profile page on the recommender systems. The statistic section in the profile page, contains information about the reputation of services, its overall reputation, the number of requests (use) to services, the detailed feedback about its services, etc.

– Gets notification: The provider may get notifications (messages, E-mail) from the WS recommendation system when the reliability of one of its services or its QoS parameters are low then expected.

4.3.2 Designer

The designer (or the stakeholder in general) of web services orchestration interacts with the WS recommendation system, to select reliable web services. The designer contributes on enhancing reputation evaluation by monitoring its running services orchestration, and then sending QoS and feedback ratings to the WSR system. Thus, the WSR system can calculate reputation values of services which enables better service recommendation.

To assist the provider in achieving the previous goal, we propose that she/he set up a monitoring application with a 'Reliability Checker and QoS feeder', 'Event manager' and a designer interface(See figure 4.2). The 'Monitor' component permits an accurate watching of services orchestration. The 'Reliability Checker and QoS feeder' sends a detailed feedback about monitored QoS values to the WSR system. In addition it verifies the reliability and other QoS parameters based on the SLA fixed initially.

The 'Event Manager' handles incoming events. It should be configured to receives WSRS events. These events are messages or notifications about the presence of more reliable substitutes in the system. Thus, the orchestration stakeholder

The orchestration stakeholder can send feedback ratings to the WSR system, representing its satisfaction about services that she/he selected for constructing the service orchestration.

Other interactions between the orchestration stakeholder and The WSR system are summarized as follows:

– Subscribes to the system: the designer could subscribe to the WSR system by introducing an identification ID, Email and a password.

– Creates a profile for new orchestration (i.e., defines the abstract process): The WSR system allows the designer to create multiple profiles for each orchestration that she/he wants to develop. The designer specifies the abstract process of the orchestration, and details about functionalities and QoS of each element (service) in the process. The WSR system recommends services in function of the introduced specifications.

- Searches and selects candidates: The designer is also able to search by keywords for specific web services. In addition, it could search for substitutes for a given service or for specific operations. The designer could include these services in its orchestration profile.
- Compares between different selections: before orchestration binding, the designer could evaluate the overall reliability and QoS of the WS orchestration that she/he designs. Using different concretization based on the recommended services and their substitutes.
- Receives notifications: The designer can set up its profiles to get notifications when reliability of selected services is changed or when new service substitutes with high QoS are introduced to the system.

4.3.3 Web Service Recommendation System

The Web Services Recommendation system (WSR system) is the principal element in the proposed architecture. It assists stakeholders and service providers to enhance the reliability of their web services and orchestrations by providing the former by recommendations about reliable and reputable web services and their candidates. It notifies the later (the service provider) by the opinion of users on the quality of its services quantified in reputation scores. Therefore, both of them can take action to enhance the reliability of their services.

The WSR system serves as a smart registry that organizes, clusters and identifies similarity and composability relationships between services. Thus, it simplifies the selection of relevant web services by designers and developers. Moreover, the system stores QoS values of services and tracks their updates in special registry. Therefore, it can conduct reliability analysis and QoS verification, and offers these informations to its requesters.

In addition, the system manages completely reputation of web services. It collects users' feedback ratings and aggregates them into reputation scores. It evaluates also the reputation of services provider. Thus, the system is able to recommend trustful and reputable web services.

The WSR system is composed from many components that we describes in the following subsections.

1- Subscription, search and selection interface

This interface permits to service provider and clients (designer, developer, etc.) to: (1) subscribe to the system, (2) search for web services by keywords or

browsing the catalog of services, (3) allow the selection of web services. This component provides informations about QoS and reputation of web services.

2- Provider interface

This component permits to service provider to check the QoS and reputation values of its web services. It provides also informations about QoS and Reputation of concurrent web services.

3- Feedback collector

This component collects and organizes the feedback ratings of users. It tracks the list of used services, and asks their users to provide feedback ratings on how reliable was the service. The component sends collected feedbacks to the Data Manager to store them in the correspondent Database.

4- QoS collector

This component listens to feeds of the provider's and the designer's applications. It receives from their QoS feeders the results of monitored services. The component sends the collected QoS to the data manager which stores these values in the appropriate Database.

5- Event manager

This component notifies designers by the presence of new service substitutes with better reputation and Qos values. The notifications are based on designers specifications which are written in orchestration profiles.

In addition, the component notifies service providers by reliability values, reputation values and QoS changes occurred on its published web services.

6- Subscription manager

The component is responsible for creating profiles for service providers and clients. It manages their subscription and checks requirements for providing better recommendation services.

7- Reputation manager

This component measures the reputation of web services, the credibility and the reputation of their providers from the feedback rating collected from web service clients. This component implements the reputation assessment model proposed in Chapter 7.

This component continuously assesses reputations values of web services and updates the Reputation Database (RDB) accordingly through the data manager component.

8- Data manager

The Data manager component has an internal role in organizing data in appropriate Databases. It is the interface of other components to get access to data.

The data manager interacts with the following Databases:

1. **Profile Database**, or in short **ProfDB**: it holds data of users(clients, providers, designers, developers) profiles.
2. **Similarity Database**, or in short **SimDB**: It holds similarity values between web services or their operations. Service substitutes are retrieved from this database. In addition, it holds the composability relationships between services or their operations.
3. **WSDL Database** or in short **WSDLDB**: it holds the WSDL files of the published services.
4. **QoS Database** or in short **QoSDB**: it holds QoS values of web services provided from different web service monitors.
5. **Feedback ratings Database** or in short **FeedbackDB**: it holds all the feedback ratings collected from clients by the feedback collector. The system keeps all the feedback ratings even those detected as malicious feedbacks.
6. **Reputation Database** or in short **RDB**: This database holds the calculated reputation scores of web services and providers.

9- Service recommender

The role of the recommender component is to propose web services for clients answering their search queries. The recommended service results are sorts based on reputation and QoS parameters.

10- Similarity assessor

This component has an internal role; It evaluates the similarity and composability relationships between web services. It stores similarity and composability scores in the appropriate database through the data manager. Practically, this component can integrates the tool described in Chapter 5.

11- Substitutes identifier

This component identifies simple and complex service substitutes (See Chapter 6). Thus, the WSR system be able to proposes replacement for defected services or for services that do not guarantee a certain level of QoS and reputation.

12- QoS and reliability checker

This component checks that QoS and reliability requirements described in orchestration profiles are achieved by the selected services. If the component detects any violation in these requirements, it asks the event manager to send notification to the orchestration stakeholder.

4.4 Summary

In this chapter, we have presented a refined service orchestration life-cycle. We have seen that reliability requirements have to be expressed explicitly at initial phases. Therefore, orchestration stakeholder can manually or automatically conduct a reliability analysis. We have noted that runtime monitoring is an important phase that is needed to detect any reliability or QoS requirement violations. Moreover, a smart registry should recommend for orchestration stakeholder reliable service substitutes. Thus, orchestration stakeholder can update its service compositions when it is needed.

In the second part of this chapter, we have proposed an architecture that assists web service orchestration stakeholders in developing and maintaining more reliable services. The architecture is composed of three participants: (1) Service provider, (2) Orchestration designer or stakeholder and (3) Web service recommendation system. The central role in this architecture is ensured by the WSR system that assists in providing reliable services and their substitutes. We have described the composition of each participant, as we have briefly described the interactions between these elements to ensure reliable services orchestrations.

In next chapters, we present more details about similarity assessment, substitutes identification, and service reputation management which are the core component of the proposed Web service recommendation system.

Chapter 5

An approach for web service similarity assessment

Contents

5.1 Overview

Similarity measurement between web services is a key solution to benefit from the reuse of the large number of web services freely available in the Internet. It is

also a considerable factor for building compositions and healing them by finding relevant substitutes for failed web services.

In this chapter, We present a practical approach that enables an effective measurement of web service similarity based on their interfaces described within WSDL documents. The approach relies on the use of multiple matching techniques and different semantic and structural similarity metrics. The measurement of similarity determines the best substitute for a failed web service. So, it serves as a good indicator of the substitutability relation and thus of the capacity for reuse.

This chapter is organized as follows: Section 5.2 introduces the service similarity-measurement approach. Section 5.3 describes WSSIM: a support tool that implements the similarity measurement approach. Section 5.4 presents the conducted experiments that validates the proposed approach.

5.2 Similarity assessment approach

The proposed approach to measure the similarity between Web services depends on services WSDL interfaces. Thus, the elements of WSDL documents are considered by the measurement process. We limit the similarity measurement to a subset of WSDL elements. This subset includes:

- `Service`: a service element consists of a set of nested operations. It is described by a name and a textual documentation.
- `Operation`: an operation element is described by a name and a textual documentation. It contains input and output messages.
- `Message`: a message element is described by a name and eventually a textual documentation. A set of parameters are held by each message.
- `Parameter`: a parameter is described by a name and eventually a textual description. The parameter could be of simple or complex type.
- `Types`: a type element defines the data type of a parameter within a web service using some type system such XSD.

A hierarchy of functions that deals with measuring similarity between pairs of compared WSDL elements is defined as the core of the implemented approach. Each function returns a similarity score that ranges between 0 and 1; 0 means the elements are totally different, 1 means that they are totally similar. We assign weights to theses scores. By default, the value 1 is assigned to all scores.

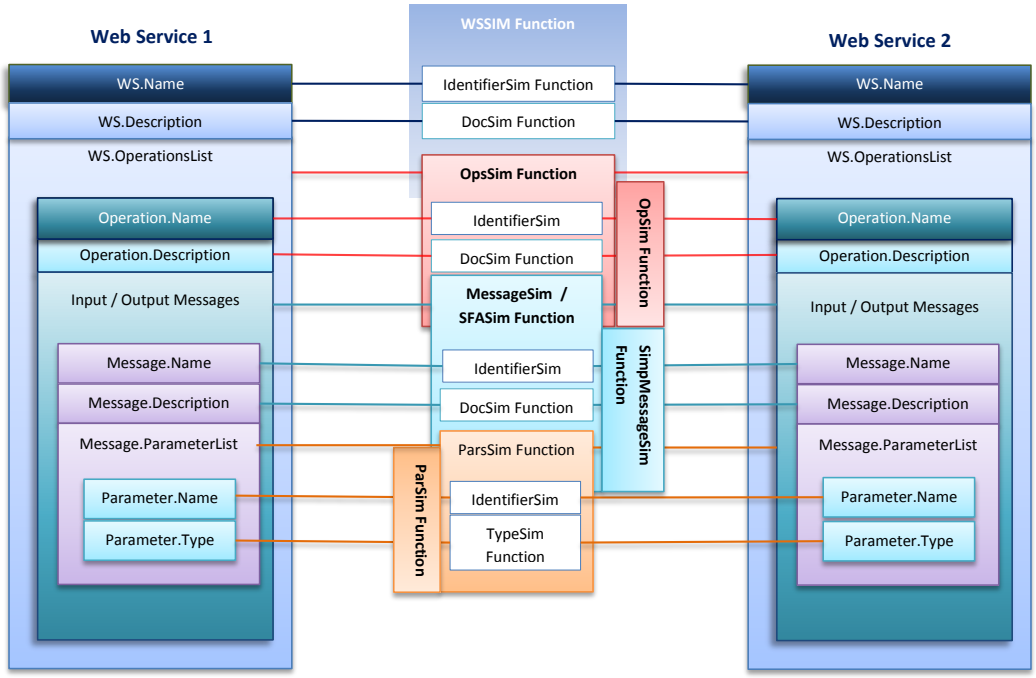

Figure 5.1: Similarity Measurement Process

The final score is calculated depending on these weighted scores. It is possible to customize the weights with different values based on the user's experience.

To measure these scores, the process starts by evaluating the following function:

$$\begin{aligned}\textbf{WsSim}(Ws_1, Ws_2) = \quad & w_{SN} \times \textbf{IdentifierSim}(Ws_1.Name, Ws_2.Name) \\ + \quad & w_{SO} \times \textbf{OpsSim}(Ws_1.OpList, Ws_2.OpList) \\ + \quad & w_{SD} \times \textbf{DocSim}(Ws_1.Doc, Ws_2.Doc) \\ / \quad & (w_{SN} + w_{SO} + w_{SD}) :\end{aligned}$$

where:

- `WsSim` is the main function which is called for measuring similarity between two web services denoted Ws_1 and Ws_2. Every Service Wsi has a name, an operation list, and a textual description denoted respectively, $Wsi.Name$, $Wsi.OpList$, and $Wsi.Doc$.
- `IdentifierSim` measures similarity between identifiers that label web services, operations, messages or parameter names.
- `OpsSim` measures similarity between two lists of operations that belong to the compared services.
- `DocSim` evaluates the similarity between two textual documentations.

- w_{SN}, w_{SO} and w_{SD} are respectively the assigned weights to `IdentifierSim`, `OpsSim` and `DocSim`.

Additionally, these functions depend in their tasks on other sub-functions. a list of sub-functions is presented bellow, and more details about these sub-functions are covered in the next subsections.

- The `OpSim` function measures similarity between a single pair of operations (subsection 5.2.4).
- The `MessageSim` function measures similarity of a single pair of messages (Subsection 5.2.5), `MessageSim` is built upon SfaMessSim and SimpMessageSim functions.
- The `SfaMessSim` function evaluates the similarity of a single pair of message elements using the Similarity Flooding Algorithm proposed in [109] (Subsection 5.2.5).
- The `SimpMessageSim` function measures the similarity between names, documentation and parameters of two compared message elements (Subsection 5.2.5).
- The `ParsSim` measures similarity between two sets of parameters. The parameters are the input or the output parameters of a message.
- The `ParSim` function is called by `ParsSim` function. It measures the similarity between two parameters of a simple type.
- The `SimTypeSim` function evaluates the similarity between two given simple types (Subsection 5.2.7).

An illustration of the measurement process is depicted in Figure 5.1. The process starts by the `WsSim` function. This function, as denoted previously, gets the similarity scores between names, textual documentations of the compared WSDL files by invoking `IdentifierSim` and `DocSim` functions respectively. Additionally, `WsSim` evaluates the similarity between the services lists of operations by calling the function `OpsSim`. `WsSim` assigns weights to these scores and returns the final similarity score between two web services.

The function `OpsSim` gets two lists of operations as input. It compares every pair of operations by calling the function `OpSim`. The similarity scores of compared pairs are stored in a similarity matrix. The problem of getting the maximum similarity score from the matrix is addressed as finding the maximum

weighted assignment in a bipartite graph. This later is implemented by the function `HungarianMax` which returns the maximum similarity score between the two lists of compared operations (more details are covered in Subsection 5.2.8).

The function `OpSim` uses `IdentifierSim`, `DocSim` and `MessageSim` to calculate similarity between two operations by comparing their names, descriptions, input/output messages.

The measurement of similarity between messages (input messages with input messages and output messages with output messages) is assigned to the function `MessageSim`. This function measures the similarity using two methods: 1) Measuring the similarity using the algorithm proposed by SFA . 2) Measuring the similarity based on message signature matching. The first method is implemented by the function `SfaMessSim`, and the second one is implemented by the function `SimpMessageSim`. The `MessageSim` function returns the maximum score of the two results.

Likewise to `OpsSim`, `ParsSim` evaluates the similarity between two lists of parameters (used in `SimpMessageSim`). The similarity between each pair is calculated by the function `ParSim`. The results of all pairs are represented as a similarity matrix. The `HungarianMax` function uses the similarity matrix to return the maximum similarity score between the two parameter lists.

5.2.1 Identifiers similarity

In WSDL document, an identifier is a unique or a sequence of concatenated words that identifies a web service, an operation, a message or a parameter. The function `IdentifierSim` deals with the measurement of similarity between two identifiers. This function could even be used in the context of comparing databases or XML schema, software models, or portion of codes.

The function `IdentifierSim` measures the similarity between two identifiers in respect to the following steps :

1. **Tokenization**: tokenization is the task of chopping up an identifier into pieces (words), called tokens. So, two sets of tokens are generated. Each one corresponds to an identifier. Stop words are removed from the two sets.

2. **Tree Tagging**: the tree tagging aims to annotate the extracted tokens with their grammatical position in the whole identifier.

3. **Similarity Matrix Generation**: A similarity matrix is generated based

Algorithm 1 Word_similarity_assessment_algorithm

Input: $Word_1$, $Word_1$, WordNet, SemanticMetricSet, StructureMetricSet
Output: *SimScore*
Begin

if ($Word_1 \in$ WordNet) && ($Word_2 \in$ WordNet)) **then**
    SumScores = $\sum_{i=1}^{n}$ Sim($Word_1$, $Word_2$, $metric_i$), $metric_i \in$ SemanticMetricSet;
    $SimScore = \frac{SumScores}{n}$;
else
    for all $metric_i \in$ StructureMetricSet **do**
        ASimScores[i]= Sim($Word_1$, $Word_2$, $metric_i$) ;
    end for
    DecendingSort (ASimScores) ;
    SumScores = $\sum_{i=1}^{m}$ AsimScores[i];
    $SimScore = \frac{SumScores}{m}$;
end if
Return *SimScore* ;

End

on the similarity of tokens-tuples. Each cell in the matrix holds a similarity between two compared tokens. Where tokens are picked from the two sets. So a line in the similarity matrix corresponds to the similarity scores between a token from the first set with all tokens in the second set. Structural and semantic metrics are involved in the computation of the similarity scores stored in matrix cells. Algorithm 1 describes how appropriate metrics are selected to measure the similarity between two words (tokens).

4. **Maximum similarity score assessment**: The first step in this task is to select the best similarity scores between token-tuples from the similarity matrix. Each token figures only once in the selected tuples. Then, the average of these maximum scores is returned as the final similarity score between the compared identifiers. (Selection and computation are explained in more details in Section 5.2.8).

5.2.2 Documentation similarity

As a matter of fact, available web services do not contain full description/documentation of all WSDL elements. Hence, we ignore evaluating sim-

Table 5.1: List of similarity metrics

Structure Metrics
Stoilos [151]
ChapmanOrdName [29]
Jaro [78]
JaroWinkler [181]
Levenshtein [92]
NeedlemanWunch [118]
QGramsDistance [171]
SmithWatermanGotoh [149]

Semantic Metrics [132]
Jiang
Lin
Pirro Seco
Resnik

ilarity between documentations once they are missed from the WSDL files. Otherwise, we compare the textual descriptions (documentations) using `LSI` [49] and `TF/IDF` [13] measures which are widely used in information retrieval [184]. The function DocSim based on these measures evaluates and returns the similarity between two compared documentation elements.

5.2.3 Grammatical tags for enhancing identifier similarity

In order to consider grammatical aspects during the similarity assessment between identifiers, a second version of the function `IdentifierSim` has been developed. This later uses the Tree tagging technique. Tree-Tagging consists of annotating text by part-of-speech (POS) and lemma information based on both word definition, as well as word context.

In the identifiers similarity context, the generated tags (Noun, verb, Adjective, Proposition, etc.) assigned to the identifiers-tokens are used to affect similarity values which are stored in the similarity matrix. Therefore, the similarity scores between tokens are kept as calculated if the tokens have the same generated tag. And, similarity values are lessened to the half if their associated POS tags are different. As an illustrative example, the similarity between the identifier `"GetWeatherByPlaceName"` and `"GiveWeatherByZipCode"` is assessed as follows:

1. Tokenization :

   - GetWeatherByPlaceName : Get, Weather, By, Place, Name
   - GiveWeatherByZipCode :give, Weather, By, Zip, Code

Tokens	Get	Weather	Place	Name
Give	0,58	0,25	0,16	0,22
Weather	0,47	1	0,25	0,23
Zip	0,37	0,22	0,32	0,27
Code	0,37	0,24	0,49	0,47

(a)

		VB	NN	NN	NN
	Tokens	Get	Weather	Place	Name
VB	Give	0,58	0,12	0,08	0,11
NN	Weather	0,23	1	0,25	0,23
NN	Zip	0,18	0,22	0,32	0,27
NN	Code	0,18	0,24	0,49	0,47

(b)

Figure 5.2: Identifier similarity matrix sample

During stop word removing the token "By" will be dropped from the two sets.

2. Tree Tagging : the result of tree tagging is :
   - get/VB weather/NN place/NN name/NN
   - give/VB weather/NN zip/NN code/NN
3. Similarity matrix generation: Figure 5.2 depicts the similarity matrix of the compared tokens. The initial matrix (a) groups similarity values between tokens without taking into account their POS tags. And the final matrix (b) is the transformation of (a) after including POS Tags.
4. The maximum similarity score between the two identifiers is AVERAGE(0.58+ 1+0.49+0.27) =0.59.

5.2.4 Operations similarity

Measuring similarity between two operations is based on similarities between their names, descriptions and the input/output messages. The `OpSim` function handles this task according to the following formula:

$$\begin{aligned}\textbf{OpSim}(Op_1, Op_2) = \quad & w_{ON} \times \textbf{IdentifierSim}(Op_1.Name, Op_2.Name) \\ + \quad & w_{OM} \times \textbf{MessageSim}(Op_1.InMessage, Op_2.InMessage) \\ + \quad & w_{OM} \times \textbf{MessageSim}(Op_1.OutMessage, Op_2.OutMessage) \\ + \quad & w_{OD} \times \textbf{DocSim}(Op_1.Doc, Op_2.Doc) \\ / \quad & (w_{ON} + 2 \times w_{OM} + w_{OD}) :\end{aligned}$$

Where:

- Op_1 and Op_2 are the compared operations. Every operation Op_i has a name, description, input message and output message denoted respectively $Op_i.Name$, $Op_i.Doc$, $Op_i.InMessage$ and $Op_i.OutMessage$.
- w_{ON},w_{OM} and w_{OD} are weights associated respectively to `IdentifierSim`, `MessageSim` and `DocSim`.

`OpSim` is also used by `OperationsSim` where it generates the score of all operations. This is done by retrieving the maximum score from the operations similarity matrix. Cells of the matrix hold the result of `OpSim`. The maximum score is computed according to the function presented in Section 5.2.8.

5.2.5 Messages similarity

A SOAP message outlines the input or the output of an operation in a WSDL file. The message is represented in the WSDL by a name, a short description and a list of parameters. The parameters might have of a simple or a complex type. To measure the similarity between two SOAP messages, two different methods are used by the function `MessageSim` which returns the final similarity score. The first method is implemented by the function `SfaMessSim` on the basis of the similarity flooding algorithm [109], and the second method is implemented by the function `SimpMessageSim` on basis of signature matching. The function `MessageSim` returns the maximum score of the values returned by the previous functions. The function `MessageSim` is defined as follows:

$$\begin{aligned}\textbf{MessageSim} \quad = \quad & \max(\ \textbf{SfaMessSim}(Message_1, Message_2), \\ & \textbf{SimpMessageSim}\ (Message_1, Message_2))\end{aligned}$$

where $Message_1$ and $Message_2$ are the compared messages.

- The function `SfaMessSim` is an implementation of the Similarity Flooding Algorithm. The algorithm matches between labeled oriented graphs and find similar nodes in the compared graphs. In our context, we transform a message signature into labeled oriented graph (see the example illustrated

in Figure 5.3). Then, we write down initial mapping (similarities) values between nodes. The algorithm works upon the graph and the initial mapping to compute final scores between graph nodes based on similarities of their neighborhood. Finally, the score between the message nodes is returned.

- The function `SimpMessageSim` is based on signature matching where the similarity score is computed by their names, documentations, input and output messages. `SimpMessageSim` is defined as follow :

$$\begin{aligned}\textbf{SimpMessageSim}(Message_1,\ Message_2) = \\ & w_{MN} \times \textbf{IdentifierSim}(Message_1.Name, Message_2.Name) \\ & + w_{MP} \times \textbf{ParsSim}(Message_1.ParsList, Message_2.ParsList) \\ & + w_{MD} \times \textbf{DocSim}(Message_1.Doc, Message_2.Doc) \\ & /(w_{MN} + w_{MP} + w_{MD})\end{aligned}$$

Where:

- $Message_1$ and $Message_2$ are the compared messages. Every $Message_i$ has a name, documentation and a list of parameters denoted respectively $Message_i.Name$, $Message_i.Doc$ and $Message_i.ParsList$.
- w_{MN}, w_{MP} and w_{MD} are weights assigned to the different used functions.

It is important to note that we can also study similarity between input messages with output messages in order to detect eventual composition possibilities. This is out of the scope of this paper which deals with substitution and not composition.

5.2.6 Complex-type parameters similarity

The measurement of similarity between complex parameters is a challenging problem. In addition to the use of similarity flooding algorithm, we solve the problem of complex-types comparison by breaking complex types into a set of simple parameters (set of sub-elements). The following steps describe how to measure similarity between complex-parameters:

1. Transform complex parameters to a set of simple parameters: In this step, complex parameters are replaced by their simple-type parameters (sub-elements). Where, the identifiers of the subelements are aggregated with the identifier of the parent element.

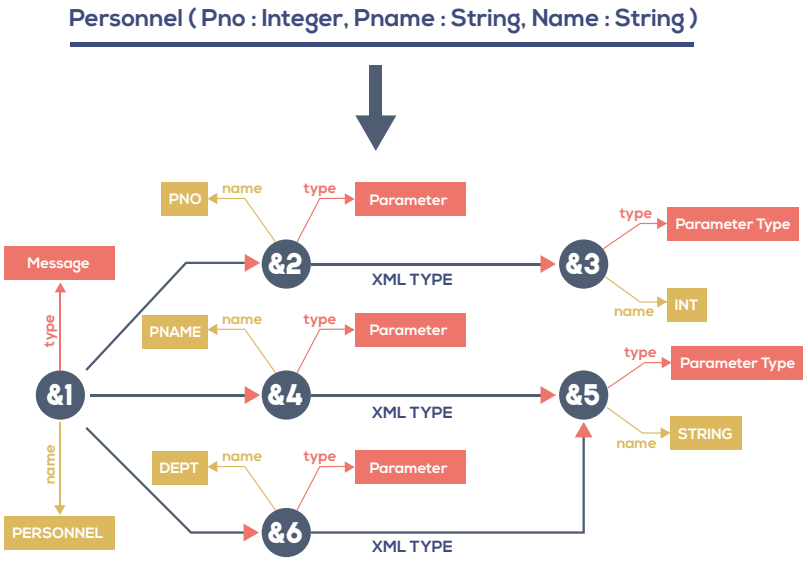

Figure 5.3: An oriented labeled graph representation of a sample Message

2. Generate the matrix of parameters similarity: `ParsSim` takes the output of the last step to generate a similarity matrix. The cells of the matrix contain scores of each parameter tuple. These similarity scores are extracted using `ParSim` (see section 5.2.7).

3. Calculate the maximum score from the similarity Matrix: see the algorithm detailed in Section 5.2.8.

5.2.7 Simple-type parameter similarity

Considering similarity between simple types as the average between their name and type similarities, Name similarity is calculated using `identifierSim`, while Type similarity is implemented using the solution proposed in [152] and [133]. Similar types are grouped in five categories. Similarities between the groups is presented in Table 5.2.

The function `ParSim` computes the similarity between two simple types. It is defined as follows:

$$\textbf{parSim}\,(Parameter_1, Parameter_2) = IdentifierSim(Parameter_1.Name, Parameter_2.Name) + TypeSim(Parameter_1.Type, Parameter_2.Type)/(2)$$

Where:

Table 5.2: Similarity table between dataType [133]

	Integer	Real	String	Date	Boolean
Integer	1.0	0.5	0.3	0.1	0.1
Real	1.0	1.0	0.1	0.0	0.1
String	0.7	0.7	1.0	0.8	0.3
Date	0.1	0.0	0.1	1.0	0.0
Boolean	0.1	0.0	0.1	0.0	1.0

Table 5.3: Simple dataType groups [133]

Group	**simple XSD Data types**
Integer Group	Integer, byte, short, long
Real Group	real, float, double, decimal
String Group	string, normalizedString
Date Group	date, dateTime, duration, Time
Boolean Group	Boolean

- $parameter_1$ and $parameter_2$ are the compared parameters. Each parameter $Parameter_i$ has a name and a type denoted respectively $Parameter_i.Name$ and $Parameter_i.Type$.
- `typeSim` evaluates similarity between two simple data types. It is obvious that the omitted weights in the function parSim are equal to 1. Because we think that the name of a simple parameter and its type are equals in importance for similarity scoring computation.

5.2.8 Maximal score computation from the similarity matrix

In order to retrieve the maximum score from a similarity matrix, the `HungarianMax` function deals with the problem as finding the maximum mean of weighted assignment in a bipartite graph. Matrix cells are considered as the edges of the graph. A match is a subset of edges where no two edges in the subset share a common vertex. In other words, it is a set of values in the matrix where no two values are from the same line or column. The assignment consists of finding the best match in the graph where each node in the graph has an incident

Table 5.4: An excerpt of a similarity matrix

	OP1'	*OP2'*	*OP3'*
OP1	0.3	**0.7**	0.9
OP2	**0.4**	0.2	0.3
OP3	0.1	0	**0.4**

edge in the match. In the matrix, the best assignment represents the maximum average of each pair of scores (line-column). Since The hungarian method [90] solves the assignment problem, it was implemented in `HungarianMax` to return the similarity score from a similarity matrix.

To illustrate the logic of this function, let us suppose that we get a similarity score between operations as depicted in Table 5.4. The maximization function returns the maximum mean score. Thus, the better bipartite matching is (OP1 - OP'2 [0.7]), (OP2 - OP'1 [0.4]) and (OP3 - OP'3 [0.4]), which equals to (0.7+0.4+0.4/3)=0.5. Eventhough, the naive composition is (OP1-OP'3 [0.9]) , (OP2-OP'1 [0.4]) and (OP3-OP'2 [0]) with scores of ((0.9+0.4+0)/3)=0.433

5.3 WSSim: a tool for measuring web service similarity

WSSim is a Java-based tool implementing the approach presented in the previous sections. *WSSim* is available as a stand-alone application, a Java API and as a web service. When paths to the desired Web services are given to the tool, it starts the assessment process by parsing the WSDL documents. Then, it calculates similarities between WSDL elements. And finally, it returns the final similarity score between the compared Web services. During the process of assessment the tool keeps similarity scores between operations, messages, and their parameters.

5.3.1 Overview of WSSim Functionalities

A screenshot of the tool is shown in Figure 5.4. In the left side, there is the list of metrics used to calculate similarity. The application of these metrics is left for manual selection. Weights are customized based upon the user experience (for example, one can put 0 for the documentation similarity, because she/he does not trust on the documentation provided in WSDL documents). There is a manual evaluation of importance of some functions using weights (for example,

similarity between input/output messages of operations is more important than similarity between names. In another case, one can consider parameter names more important than their types or *vice versa*). There are different tabs for viewing details about similarity scores once extracted (see the enlarged squares in Figure 5.4).

The similarity for substitution is viewed by WSSim by giving some suggestions of the operations of other web services that best much a given Web service operation. This is illustrated in the bottom-left corner of the Figure.

5.3.2 Underlying Technologies

The following APIs have been used to develop this tool:

- *SFA API* [109]: This API is a Java implementation of the Similarity Flooding Algorithm (found in: `http://infolab.stanford.edu/\protect$\relax\sim$melnik/mm/sfa/`). In our tool, we use the SFA API to compute similarity between two messages. The RDF model of the two messages is generated by *WSSim* before calling the API.
- *WordNet*[1]: It is a lexical database for English words. Words in the database are grouped into sets of synonyms called synsets. *WSSim* uses WordNet to find semantic relations between compared words. It is also implicitly used with the semantic metrics in order to evaluate the similarity between names.
- *SimMetrics*[2]: It is an open source Java library of similarity metrics between strings. All metrics in the library can work on a simple basis taking two strings and returning a measure from 0.0 to 1.0. The library is used in *WSSim* in order to evaluate structural similarity between words. The used metrics are listed in Table 5.1.
- *JDOM*[3]: It is a Java API for processing XML documents. It is used to parse WSDL files. The parsing consists of extracting web service elements and representing them as a basic object model.
- *JWS*: API library for semantic similarity measurement based on WordNet. The library is developed by *Pirro and Seco* [132].
- *Stanford PosTagger*: a library for Part-Of-Speech Tagging[4] developed by

[1]WordNet: `http://wordnet.princeton.edu/`

[2]Open source Similarity Measure Library: `http://sourceforge.net/projects/simmetrics`

[3]JDOM: `http://www.jdom.org`

[4]Stanford POS Tagger API: `http://nlp.stanford.edu/software/tagger.shtml`

Stanford NLP group.

The tool offers an open-source user-friendly interface and an API. It is designed to be flexible for both simple users and third-party developers. *WSSim* is available on the following link: https://code.google.com/p/wssim/

5.3.3 WSSim as a Web Service

A version of WSSim is also available as web service in order to ease its use by third-party developers. The web service groups three operations; (i) the *getServiceSimilarity* operation which returns the overall similarity score between two services, (ii) the *getOperationPairInfo* that returns information about similarity, substitutability and composability between operation pairs, and (iii) *getSubstitutableOperations* that returns the operation-pairs considered substitutable by the tool.

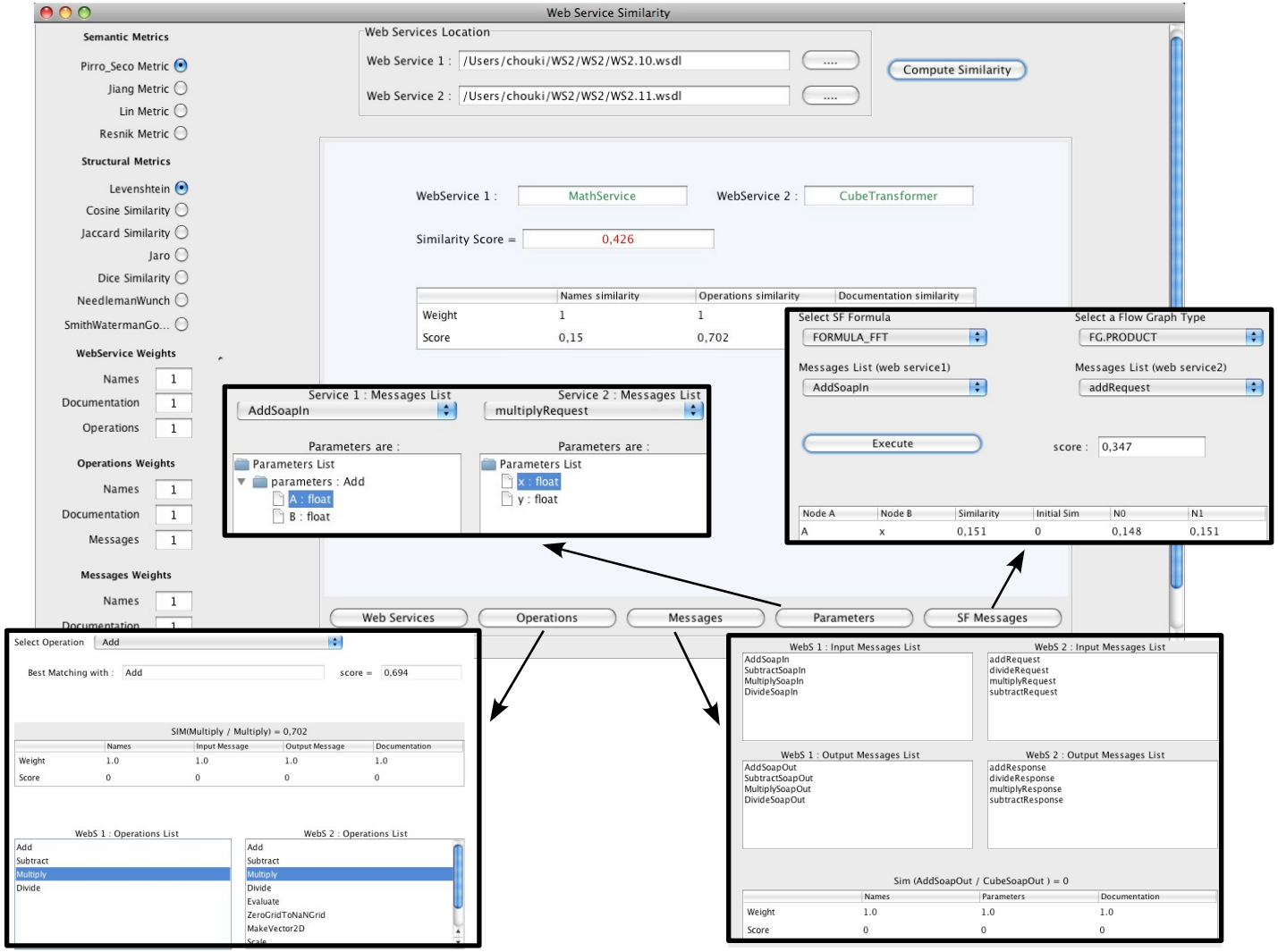

Figure 5.4: Screenshot of the tool WSSim

5.4 Experiments and validation

The approach and its support tool have been experimented on real-world Web services. Unfortunately, we were not able to test the implementation of similar tools to compare their results against those generated by WSSim. Nevertheless, we run several functional tests to obtain more consistent results according to a human evaluation.

5.4.1 Tuning

At the beginning, we ran many tests to definitively fix the set of similarity metrics used by WSSim (The final set is listed in table 5.1). In addition, we compared results obtained when the identifier similarity function uses the tree tagging technique with those obtained without using the tree tagging technique. The collection of identifiers used for test was extracted from real web services.

Reported results were compared against a human evaluation of similarity of the executed test cases. Figure 5.5 depicts an excerpt of compared identifiers used during this first step.

As an example, `WSSim` returns 0.833 between two given Identifiers: GetUniversityName and GetCollegeName. Since we consider a similarity score that ranges between 0.80 and 1 as high, the human evaluation of the test case has confirmed it.

The results also had shown that the use of Tree Tagging enhances the similarity between identifiers in some cases. But generally, the results obtained by the function without Tree Tagging are close to those obtained by the use of Tree Tagging.

5.4.2 Case Study

In order to check the effectiveness of the approach and its implementing tool, a case study is conducted to evaluate similarity between a set of real web services, and to find relevant substitutes for service operations depending uniquely on similarity scores. The experiment has been conduction following these steps:

Collecting WSDLs

we were interested to study similarities between real web services offering IP information, ZipCode Information and Weather Information. Thus, we used the keywords "IP", "ZIP" and "Weather" to find corresponding WSDLs. We retrieve

Identifier 1	Identifier 2	Similarity by the tool		Manually evaluated
		With TreeTaggin	Without TreeTagging	
CurrencyExchange	MoneyExchange	0,8979194	0,9063004	**Very High**
getMsgId	getMessageIdentifier	0,91411704	0,91411704	**Very High**
getPersonIdentifier	getHumanId	0,8641388	0,8641388	**Very High**
getWeatherByCityName	getWeather	0,8614391	0,8614391	**High**
getWeatherInState	getWeatherByCityName	0,6666667	0,51347536	**High**
getUniversityName	getCollegeName	0,82418513	0,8332458	**High**
getWeatherByPlaceName	getWeatherByZipCode	0,54157954	0,54157954	**Medium**
getFlightBySourceAndDestination	getTravelByCityNames	0,52853566	0,56327814	**Medium**
getTempurature	getWeatherByCityName	0,22764647	0,28714636	**Low**
getScore	getScale	0.32280523	0,32280523	**Low**
getWeatherinState	ConvertCurrency	0,39628065	0,23751307	**Very Low**

Figure 5.5: Similarity results between a set of compared identifiers

Keyword	Number of Services	File Suffix
ZIP	20	Z
Weather	20	W
IP adress	20	IP

Table 5.5: Number of retrieved web services

services from ServiceXplorer[5], Service Repository[6], XMethods[7] and the service collection of OWLS-TC [8]. We selected 60 web services corresponding to the previous keywords (see Table5.5).

First run and WSDL filtering

After grouping the WSDL files in the same directory, we run the first experiment on that group. The tool detected 9 duplicate WSDL documents, even if these WSDL have different names and different extensions (wsdl, asmx, xml) and retrieved from different sources. The tool returned the similarity value 1 for the totally similar services. We filtered these services and we kept one copy of each service.

Second Run and System performances

WSSim offers the possibility to compare a group of WSDL documents and returns the similarity matrix between all services. Additionally, it returns all similar

[5]ServiceXplorer: http://eil.cs.txstate.edu/ServiceXplorer/results.php
[6]Service Repository: http://www.service-repository.com
[7]XMethods: http://www.xmethods.com/ve2/index.po
[8]OWLS-TC: http://projects.semwebcentral.org/projects/owls-tc

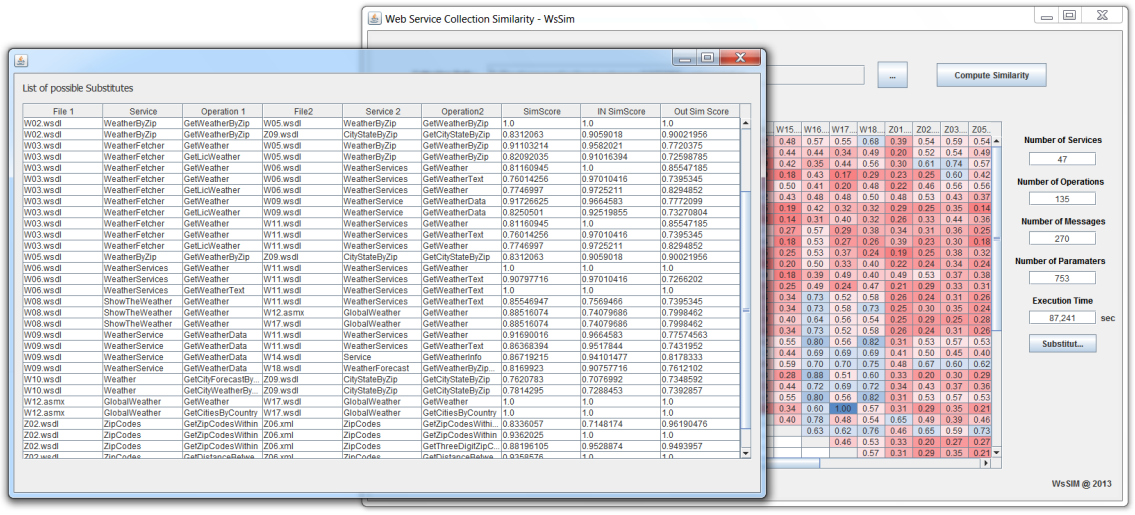

Figure 5.6: Substitution results

operation-pairs with their similarity score, Input messages similarity and output message similarity scores.

For substitution, the tool returns a list of substitutable operations as depicted in Figure (5.6). The tool select all Operation-pairs with similarity score greater than 0.7, and message similarity score greater or equal to 0.75. Operation pairs that do not satisfy the previous criteria are considered not substitutable.

The machine used in the experiment run with Intel processor (I3-2100 CPU 3.10 GHZ), and RAM memory of 4 GB with Windows 7 as an operating system. As depicted in Figure(5.6), The tool took only 88 seconds to parse 47 Web services and measures similarities between 135 operation (135*134), and 270 messages (270*269) with 753 Parameters.

Human evaluation of operations pairs

To check the accuracy and the effectiveness of the automatic selection of operation-substitutes resulted by the tool, we conduct a manual evaluation of the similarity between operations. All operation-pairs with similarity score greater or equal to 0.5 were verified. We consider two operations as substitutable if they have a similar identifier and they have the same input and the same output messages even with some adaptation (ex. parameter casting).

Criteria	
Operation Similarity	>=0,7
Input Message Similarity	>=0,75
Output Message similarity	>=075

TP (true positive)	FP (false positive)
33	**14**
FN (false negative)	TN (true negative)
18	**756**

Accuracy	**0,961**
Precision	**0,702**
Recall	**0,647**
F1-Score	**0,673**

Figure 5.7: Experiment Results

Result analysis

The results of the manual annotation (substitutable or not substitutable) were compared against WSSim results. Figure (5.7) depicts experiment results with the associated Precision, Recall, Accuracy and F-Score.

The number of False Negatives Influenced the System Recall. False Negatives in this experiment are the operation-pairs considered by the tool as non-substitutable, and the human evaluation shows that these pairs can be considered as substitutable after adaptation at the output message level. After a manual checking we observed that the tool failed in detecting the similarity between these pairs because of the comparison between a simple parameter type (String) in the output message of one of the operations, and the complex parameter type in the output message of the other operation.

Also, the number of true negatives is important and this is natural because most operations in different services are not similar.

5.5 Summary

In this Chapter, we have proposed a practical approach for measuring the similarity between Web services by comparing their interface descriptions (WSDL documents). The approach is based on the use of a set of existing lexical and semantic metrics. The measurement process is parametrized by a collection of weights associated to the different levels of web service description. The challenge of measuring the similarity between complex types, which are generally represented by XML schema, is handled by using different techniques for getting the

best scores as described previously. Obviously, the need for similarity assessment is generally adapted for composition and substitution; by finding similar services or similar operations, we can replace failed services/ failed operations by similar ones. Also, it is possible to compose from several operations, which have similar input-output messages, an equivalent failed operation (Op_{failed}=Op_1+...+Op_n).

In addition, we have presented a prototype tool (coined WSSIM) that implements the similarity measurement approach. The tool is developed using Java programming language and a set of APIs. It can measure the similarity between pair of web services, as it can measure the similarity between a collection of web services. The tool is customized by different weights, and it can evaluate the similarity between each pair of WSDL elements.

The tool has been experimented on a set of real-world Web services to show its accuracy, efficiency and practicability.

In the following chapter, we present an approach that includes the proposed similarity technique in a process that identifies services substitutes.

Chapter 6

Web Service Substitutes Identification Approach

Contents

6.1 Overview

This chapter covers the proposed process for substitute identification. First, we give an overview about the proposed process. Then, we present an example that we use for illustrating the application of the identification process. Finally, we describe in full details, within the remaining subsections, the different components of the process.

6.2 Case study

We use for illustrating the substitute identification process, a weather widget example. The widget uses an orchestration of web services to show weather information of the user's region based on its IP address. The widget interacts mainly with the web service WWS (*WidgetWeatherService*). This web service itself invokes an orchestration of two services GWP (*GettingWeatherProcess*); the WU (*WsUsers*) web service that affords IP information, and the FW (*FastWeather*) web service that returns weather information.

Figure 6.1 shows the BPEL abstract description associated to this example. Now, let us assume that the invocation of the operation marked with a red cross in Figure 6.1 (*getWeatherByIP* operation) initiates an error due to the unavailability of the service FW. This situation directly leads to the defection of the orchestration GWP. One possible solution, to heal this orchestration, is to find a substitute for the failed service (FW). Then, we reconstruct the orchestration using the identified substitute. In the next subsections, we use the weather widget as a running example for illustrating our process' components.

6.3 Architecture's overview

The substitute identification approach is built upon two techniques; the first is the similarity assessment between web services that enables to evaluate similarity and composability relationships between a set of web service candidates; and the second technique is FCA, which is used for classifying the compared services to retrieve substitutes. The identification process is depicted in Figure 6.2. It starts by analyzing the WSDL document of the failed web service by the first component (Component 1 in Figure 6.2). This component extracts, from the WSDL specification a set of representative keywords and the signature of the concerned operation. Then, the service retriever (Component 2 in Figure 6.2) uses the set of keywords to select from a web service pool, all possible similar services, i.e. it selects web services that hold at least one of the keywords and may offer the same or related functionalities. Next, the service filterer (Com-

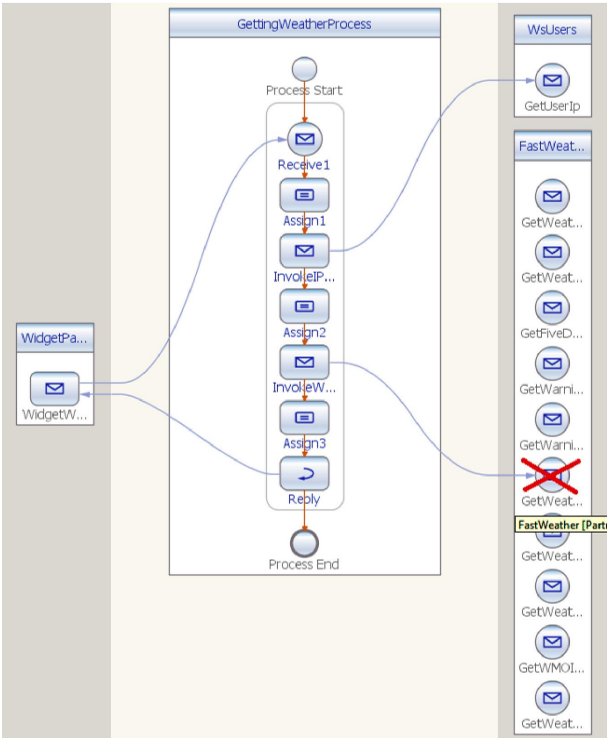

Figure 6.1: Abstract BPEL description for the weather widget example

ponent 3 in Figure 6.2) analyses the retrieved service candidates. It compares, trough similarity assessment, the operation signatures of these services with the signature of the failed one. The component keeps in the filtered service set only services that have operations which are similar to the failed service, or services that have a composability relationship with the similar services.

The next step is the construction of the similarity matrix between the elements of the filtered set and the failed service. Hence, Component 4 in Figure 6.2 assesses the similarity scores between all operation inputs and outputs (messages). The generated similarity matrix allows the identification of similarity relationships between all operations. Indeed, the similarity matrix can reveal simple substitutes. Nonetheless, the classification of operations in related groups can reveal also hidden similarity dependencies between these operations, which enables an identification of complex substitutes. For this end, the approach uses the FCA technique.

The context builder and the FCA classifier (Component 5 in Figure 6.2) transforms the similarity matrix into a formal concept after many adjustments on the matrix itself. First, it groups similar operations in clusters, which reduces the size of the matrix and enhances, at the end of the FCA classification, the

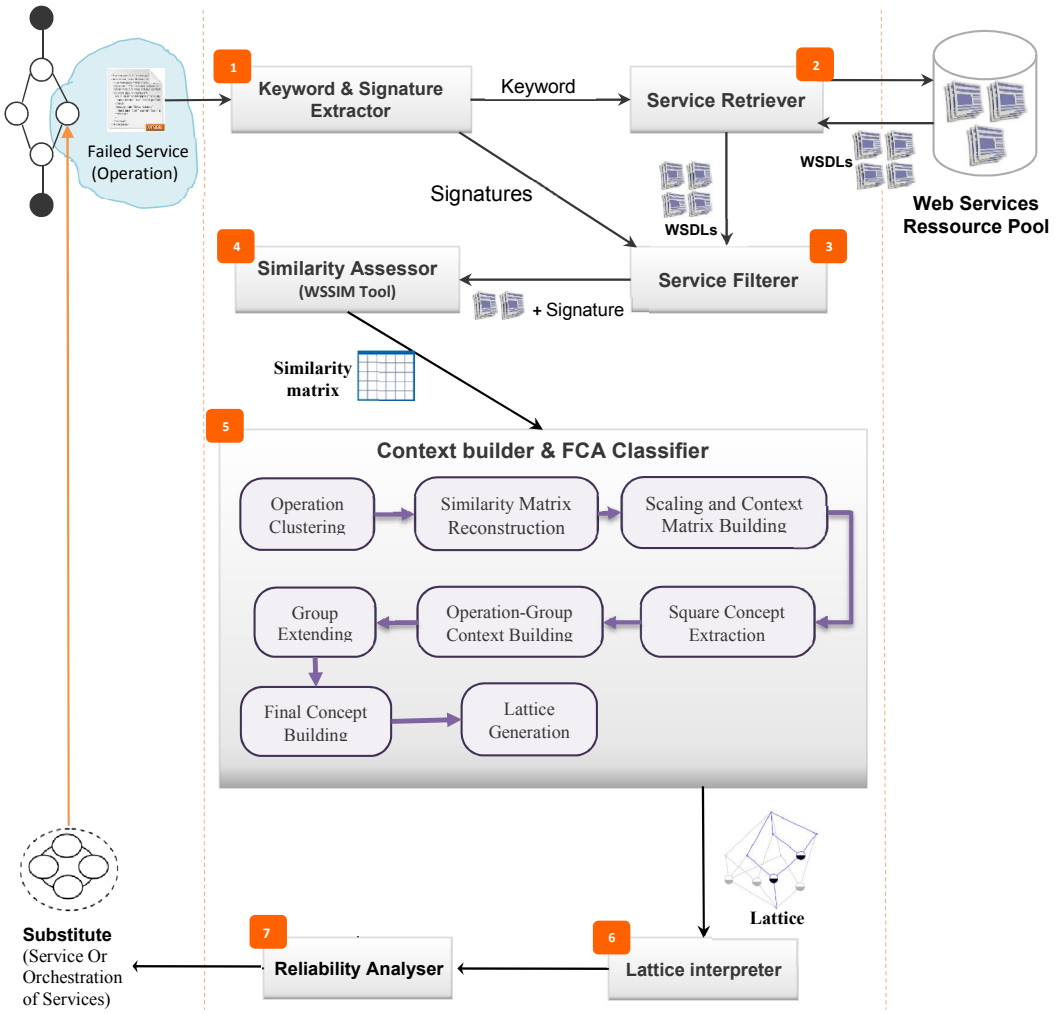

Figure 6.2: Global schema of the substitute identification process

visualization and the interpretation of the generated lattice. Second, it rebuilds the similarity matrix based on the identified groups. Next, the component builds the context matrix corresponding to the reconstructed similarity matrix. After that, the FCA classifier analyses the context matrix to select square concepts. In fact, square concepts are concepts with equal intention (operations input and output messages) and extension (operations input and output messages) sets. These concepts allow the component to construct another context called the operation-group context. In this context, the extensions are the service operations (or clusters representatives), and the intentions are groups of operations identified as square concepts. Afterwards, the component extends these intentions by adding similarity relationships to the elements of each group. Finally,

the component builds the final concept matrix and generates its corresponding operation lattice.

The lattice interpreter (Component 6 in Figure 6.2) browses the lattice and interprets its content. It focuses on analyzing the extension of the failed operation to extract the list of simple and complex substitutes.

The Reliability Analyzer (Component 7 in Figure 6.2) evaluates the reliability of the identified substitutes, and then selects the more appropriate substitutes.

6.4 Components description

This section describes functionalities and roles of each component in the architecture depicted in figure 6.2.

6.4.1 Keyword and Signature extractor

The identification process starts from Component 1 in Figure 6.2. The component parses and analyses the WSDL interface description of the failed web service to extract a set of keywords. These keywords represent semantically the functionalities implemented by the web service. The set of keywords are used as criteria for the selection of web service candidates that may offer equivalent functionalities. Moreover, the component extracts the signature of the failed operation. Later, this signature is used for similarity evaluation and candidate filtering, where the system removes all candidates, which do not hold similar operations or do not have similarity relationships with similar operations.

Keyword Extraction

Technically, the keyword and signature extractor component retrieves the representative keyword set from the parsed WSDL document through the following steps:

- The component retrieves all the identifiers from the WSDL file. Then, it adds them to an identifier set (*IdSet*).
- The component tokenizes the identifiers of the *IdSet*. It adds the extracted tokens to the Keyword Set (*KeywordSet*).
- The component treats the *KeywordSet* as follows :
  - It removes all redundant words.
  - It removes all stop words.

- It stems words.
- Finally, it enriches the set by adding words synonyms.

Thus, the component produces a set of keyword (*KeywordSet*) that represents the failed web service.

Signature extraction

Component 1 extracts also the signature of the failed operation after parsing the whole WSDL document. The signature is produced in textual format that includes operations identifier and the signature of its input and output messages. By its turn, the signature of messages holds their identifiers and a list of simple and complex parameters.

6.4.2 Service retriever

The service retriever (Component 2 in Figure 6.2) uses the elements of the keyword set to search for web service candidates that may offer the same functionalities of the failed service. The component retrieves these candidate services by either seeking available resource pool using Algorithm 2, or directly by requesting web service engines such as Service Xplorer[1].

Algorithm 2 takes as input the initial *keywordSet* (extracted from the failed service by Component 1), and a set of available services in the service pool (*ResPoolSet*). The algorithm analyzes each web service in the pool. If an analyzed service do not hold any keyword similar to those in the keywordSet, the service is shifted to the analyzed service set (*examinedServSet*). Otherwise, the service is shifted to the selected service set (*selectedServSet*), and its keywords are added to the *keywordSet* (lines 14-16). A re-analysis of the previously analyzed but not selected services is necessary, because the new added keywords could belong to one of the analyzed services. Consequently, the elements of the *examinedServSet* are shifted back to the *ResPoolSet* (lines 18-22). Finally, the algorithm returns a set of selected service candidates.

Note that within the returned service set, we have at least one service that shares some keywords with the failed web service. The other services share keywords between each other, which is interpreted by the existence of dependencies (similarity or composability relationships) between them. Component 3 and 4 check and evaluate respectively these dependencies.

[1]Service Xplorer: http://eil.cs.txstate.edu/ServiceXplorer/

Algorithm 2 Candidate_Selection

Input: *KeywordSet, ResPoolSet*
Output: *SelectedServSet*
Begin

```
1: Boolean matches ;
2: Create wordSet';
3: Create examinedServSet;
4: while (ResPoolSet!=φ) do
5:     wsdl = getElement(ResPoolSet);
6:     wordSet' = Keyword_Extraction(wsdl);
7:     matches = false;
8:     for all (String word in wordSet') do
9:         if (Contains(KeywordSet, word)) then
10:            matches = true; break;
11:        end if
12:    end for
13:    if (matches) then
14:        for all (String w in wordSet') do
15:            Add w To KeywordSet;
16:        end for
17:        Add wsdl To SelectedServSet;
18:        while (examinedServSet!=φ) do
19:            wsdl' = getElement(examinedServSet);
20:            Add wsdl' To resPoolSet;
21:            DeleteElement wsdl' from examinedServSet ;
22:        end while
23:    else
24:        Add wsdl To examinedServSet;
25:    end if
26:    DeleteElement wsdl from ResPoolSet ;
27: end while
```

End

Algorithm 3 Operation_Filtration

Input: *Signature, selectedServSet, similarity Threshold θ*
Output: *FiltredServiceSet*
Begin

```
1: Boolean matches ;
2: Create examinedServSet;
3: while (selectedServSet!=φ) do
4:     wsdl = getElement(selectedServSet);
5:     parsedWsdl = wsdlParser(wsdl);
6:     matches = false;
7:     for (int i=0; i <parsedWsdl.operationCount; i++) do
8:         operationInfo = parsedWsdl.getOperation(i);
9:         if (Sim(signature.InputMessage, operationInfo.InputMessage)≥
   θ‖ Sim(signature.InputMessage, operationInfo.OutputMessage)≥ θ‖
   Sim(signature.OutputMessage, operationInfo.InputMessage)≥ θ‖ Sim(signature. Out-
   putMessage, operationInfo. OutputMessage)≥ θ ) then
10:            matches= true ; break;
11:        end if
12:    end for
13:    if (matches) then
14:        Add wsdl To FiltredServiceSet;
15:        Boolean matches' ;
16:        Create examinedServSet';
17:        examinedServSet' = Duplicates(examinedServSet);
18:        while (examinedServSet'!=φ) do
19:            wsdl' = getElement(examinedServSet');
20:            parsedWsdl' = wsdlParser(wsdl');
21:            matches'= false;
22:            for (int i=0; i <parsedWsdl.operationCount; i++) do
23:                for (int j=0; j <parsedWsdl'.operationCount; j++) do
24:                    operationInfo= parsedWsdl.getOperation(i);
25:                    operationInfo' = parsedWsdl'.getOperation(j);
26:                    if (Sim(operationInfo.InMessage, operationInfo'.InMessage)≥
   θ‖Sim(operationInfo.InMessage, operationInfo'.OutMessage)≥
   θ‖Sim(operationInfo.OutMessage, operationInfo'.InMessage)≥ θ‖Sim(signature. Out-
   putMessage, operationInfo. OutputMessage)≥ θ ) then
27:                        matches'= true ; break;
28:                    end if
29:                end for
30:            end for
31:            if (matches') then
32:                Add wsdl' To FiltredServiceSet;
33:                DeleteElement wsdl' from examinedServSet ;
34:            end if
35:            DeleteElement wsdl' from examinedServSet' ;
36:        end while
37:    else
38:        Add wsdl To examinedServSet;
39:    end if
40:    DeleteElement wsdl from selectedServSet ;
41: end while
```

End

File	Service Name	Input		Output	
		Name	Parameter	Name	Parameter
IP02.xml (IP2Geo)	IP2Geo	ResolveIp	IpAdress :String Licence : String	ResolveIp	City : String
IP07.xml (GeometryInfo)	Geometry	IPQuery	Ip : String key : String	IPQuery	Code : String
IP15.wsdl (p2LocationWebService)	IP2Loc	IP2Location	IpAdress :String Licence : String	IP2Location	City : String
W02.wsdl (WeatherByZip)	Weather	GetWeatherByZip	Zip : String	GetWeatherByZip	CityTo1ZipResult : String
w03.wsdl (Weather)	Weather	GetWeather	Zip : String	getWeatherforZipCode	getWeatherfor-ZipCodeResponse : String Array
w06.WSDL (WeatherService)	Weather	GetWeather	Zip : String	getWeatherReturn	getWeatherReturn : String Array
Z23.asmx (ZIP)	ZIPCode	CityToZipCode	City : String	CityToZipCode	CityToZipCode : String

Figure 6.3: Filtered web services information

6.4.3 Service filterer

The filterer component refines the set of web service candidates. It removes services that do not have a similarity relationship with the failed service according to the signature provided by component 1. The similarity relation is interpreted by the fact that a given service holds an operation that is similar by its input or output message to the input or the output message of the failed operation/service. Or, it is similar by its input or output message to the input or the output message of an operation/service already evaluated as similar to the failed one.

The component uses the filtration algorithm (Algorithm 3). This algorithm takes as input the signature of the failed operation (extracted by Component 1), the selected service set (retrieved by Component 2) and a similarity threshold that represents the accepted lower limit to consider two operations as similar. The algorithm keeps in the filtered services set (*filteredServiceSet*) all services that hold an operation that is similar to the failed one, or services that hold operations that are similar to operations that are similar, by their turn, to the failed one.

In order to illustrate the remaining steps, we use the weather widget example. The service selector component retrieves a set of 142 services from a total of 3792 web services including 15927 operations in the resource service pool. Then, the service filterer component reduces the number of selected services to 7. Figure 6.3 summarizes the information about the concerned operations in the filtered

		Op_1		Op_2		...	Op_n	
		Input	Output	Input	Output	⋮	Input	Output
Op_1	Input	1		S_{I1_I2}	S_{I1_O2}	...	S_{I1_In}	S_{I1_On}
	output		1	S_{O1_I2}	S_{O1_O2}	...	S_{O1_In}	S_{O1_On}
Op_2	Input	S_{I2_I1}	S_{I2_O1}	1		...	S_{I2_In}	S_{I2_On}
	Output	S_{O2_I1}	S_{O2_O1}		1	...	S_{O2_In}	S_{O2_On}
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Op_n	Input	S_{In_I1}	S_{In_O1}	S_{In_I2}	S_{In_O2}	...	1	
	output	S_{On_I1}	S_{On_O1}	S_{On_I2}	S_{On_O2}	...		1

Figure 6.4: A generic form for the similarity matrix (SimMatrix)

services while the non-similar operations are ignored. The failed service in the example is "*FastWeather*" service (the file W15 is not included in Figure 6.3), and the failed operation in the orchestration is "*GetWeatherByIp*".

6.4.4 Similarity assessor

The role of the similarity assessor component is to measure the similarity values between input and output messages that belong to the operations in the filtered service candidates. The component arranges these values in a similarity matrix. Figure 6.4 depicts the generic form of this matrix. Practically, the component uses WsSim tool[2] for the similarity measurement. However, the component is generic, and any other similarity assessment approach can be integrated.

Actually, the elements to consider for the interpretation of the similarity matrix (Figure 6.4) are the following:

- $WS = \{Ws_y | 1 \leq y \leq l\}$. WS is the filtered Web service set (filteredServSet). Ws_i is the service number i in the filtered service set.
- $OP = \{Op_x | 1 \leq x \leq n\}$. OP is the operation set. $\forall Op_x, Op_x \in Ws_y | 1 \leq y \leq l$; i.e. all the operations in the operation set are belonging to a service in the Web Service Set.
- The function $Sim(A, B)$ evaluates the similarity between messages A and B where :

[2]Available online: https://code.google.com/p/wssim/

  - A,B ∈ MS; MS = { $Op_i.InputMessage \bigcup Op_i.OutputMessage | Op_i \in OP$ }. MS is the message set that groups the input and output messages of the operations.
  - ∀ A∈ MS,∀ B ∈ MS, A = B ⇒ Sim(A,B)=1.
  - ∀ A∈ MS,∀ B ∈ MS, A ≠ B ⇒ Sim(A,B)∈ [0, 1].

- $Op_j \in$ OP and $Op_k \in$ OP are the compared operations and both belong to the operation Set.
- $S_{Ij\_Ik} = Sim(Op_j.InputMessage, Op_k.InputMessage)$, S_{Ij_Ik} is the similarity score between the input message of operation Op_j and the input message of operation Op_k.
- S_{Oj_Ik}=Sim($Op_j.OutputMessage$, $Op_k.InputMessage$), S_{Oj_Ik} is the similarity score between the output message of operation Op_j and the input message of operation Op_k.
- S_{Ij_Ok}=Sim($Op_j.InputMessage$, $Op_k.OutputMessage$), S_{Ij_Ok} is the similarity score between the input message of operation Op_j and the output message of operation Op_k.
- S_{Oj_Ok}=Sim($Op_j.OutputMessage$, $Op_k.OutputMessage$), S_{Oj_Ok} is the similarity score between the output message of operation Op_j and the output message of operation Op_k.

Moreover, we give the following definitions to figure out similarity and substitutability relations between operations from the similarity matrix.

Definition 1 (Similar Operations): we consider two operations as similar if and only if the similarity value between operation inputs and the similarity value between operation outputs; both are greater than or equal to a threshold θ'. θ' is fixed experimentally. Mathematically, $\forall Op_x \in OP, \forall Op_y \in OP$, and $x \neq y$, Sim(Op_x.inputMessage, Op_y.inputMessage)$\geq \theta$', and Sim(Op_x.inputMessage, Op_y.inputMessage)$\geq \theta$' $\Leftrightarrow Op_x \equiv Op_y$. Op_x is similar (equivalent) to Op_y.

Definition 2 (1-to-1 substitute): from Definition 1, similar operations to a failed operation are 1-to-1 substitute to this operation; If Op_f is the failed operation, $\forall\ Op_x \in OP, f \neq x$ and $Op_f \equiv Op_x$ then Op_x is 1-to-1 substitute to Op_f. These two definitions allow operation clustering and similarity matrix reconstruction which are explained in the next subsections.

For Illustration, Figure 6.5 shows the similarity matrix (*SimMat*) between the input and the output messages of the operations presented in Figure 6.3, and

	W15.I	W15.O	IP02.I	IP02.O	IP07.I	IP07.O	IP15.I	IP15.O	W02.I	W02.O	W03.I	W03.O	W06.I	W06.O	Z23.I	Z23.O
W15.I	1		0,935	0,579	0,702	0,192	0,65	0,103	0,381	0,577	0,577	0,172	0,577	0,172	0,491	0,362
W15.O		1	0,108	0,423	0,551	0,184	0,511	0,519	0,611	0,684	0,612	0,966	0,612	0,739	0,254	0,119
IP02.I	0,935	0,108	1		0,65	0,116	0,65	0,113	0,362	0,095	0,376	0,571	0,376	0,571	0,119	0,502
IP02.O	0,579	0,423		1	0,544	0,758	0,507	0,687	0,095	0,121	0,093	0,395	0,093	0,571	0,901	0,124
IP07.I	0,702	0,551	0,65	0,544	1		0,68	0,304	0,435	0,263	0,447	0,27	0,447	0,286	0,565	0,565
IP07.O	0,192	0,184	0,116	0,758		1	0,582	0,681	0,252	0,604	0,277	0,248	0,277	0,212	0,895	0,121
IP15.I	0,65	0,511	0,65	0,507	0,68	0,582	1		0,472	0,378	0,472	0,262	0,472	0,279	0,128	0,312
IP15.O	0,103	0,519	0,113	0,687	0,304	0,681		1	0,227	0,373	0,277	0,492	0,277	0,549	0,912	0,135
W02.I	0,381	0,611	0,362	0,095	0,435	0,252	0,472	0,227	1		0,983	0,417	0,983	0,463	0,547	0,789
W02.O	0,577	0,684	0,095	0,121	0,263	0,604	0,378	0,373		1	0,64	0,683	0,64	0,683	0,535	0,483
W03.I	0,577	0,612	0,376	0,093	0,447	0,277	0,472	0,277	0,983	0,64	1		1	0,465	0,382	0,756
W03.O	0,172	0,966	0,571	0,395	0,27	0,248	0,262	0,492	0,417	0,683		1	0,51	0,769	0,246	0,176
W06.I	0,577	0,612	0,376	0,093	0,447	0,277	0,472	0,277	0,983	0,64	1	0,51	1		0,382	0,756
W06.O	0,172	0,739	0,571	0,571	0,286	0,212	0,279	0,549	0,463	0,683	0,465	0,769		1	0,246	0,321
Z23.I	0,491	0,254	0,119	0,901	0,565	0,895	0,128	0,912	0,547	0,535	0,382	0,246	0,382	0,246	1	
Z23.O	0,362	0,119	0,502	0,124	0,565	0,121	0,312	0,135	0,789	0,483	0,756	0,176	0,756	0,321		1

Figure 6.5: Case study similarity matrix (SimMat))

the input and output messages of the failed operation (*getWeatherByIp*) in the service (W15). The matrix *SimMat* is an instance of the generic matrix depicted in Figure 6.4. Lines and columns in *SimMat* represent inputs and outputs of the concerned operations (each service is represented by one operation in this example). The cells of *SimMat* hold the similarity values between lines and columns (operation's input/output, input/input, output/input or output/output).

6.4.5 Context builder and FCA classifier

The context builder and FCA classifier is Component 5 in Figure 6.2. This component classifies and visualizes the operations that belong to service candidates in a lattice. This organization exploits the similarity matrix to reveal substitution relationships between operations, in a navigable way within a generated lattice. The context builder and the FCA classifier conduct many transformations depicted inside the component (Component 5, Figure 6.2). It starts by grouping similar operations in clusters, which directly identifies simple (1-to-1) substitutes because operations in the same cluster represent substitutes for each other.

Next, the component reduces the lines and columns number in the similarity matrix based on the constructed clusters, which reduces consequently the lattice size, and hence the complexity of its interpretation. Then, the component uses the reduced similarity matrix for building its associated context matrix that includes the input and output messages in addition to their operations.

Afterwards, the classifier analyses the context matrix to recognize the square concepts, i.e. the maximal collections of messages (intentions) that share the same similarity relationships with other messages (extensions). Then, the component forms groups of similar messages. Each group holds the objects (input and output messages) of the corresponding square concept. These groups maintain the similarity relationships between messages of the same group, which represent possible compositions between operation output (output message) and another operation input (input message).

In order to show these composition relationships, the component builds what we call Operation-Group context matrix. The objects in this context matrix are the operations and the attributes are the groups of messages extracted from the square concepts. In the next step, the component extends the groups of messages (the attributes of the last context matrix) by adding for each input message its output message, and for each output message its similar messages. This step is crucial before building the final context because it identifies all composition sequences (which operation could be composable with another). These composition sequences present, during the visualization of the lattice, the complex (N-to-1) substitutes.

After that, the component builds the final concepts based on the extended groups. These concepts hold the relationships between operations and their possible substitutes. Finally, the classifier generates the lattice that corresponds to the final context. This lattice classifies operations in groups that share common attributes. In this case, the attributes hold the possible substitutes for each operation. Component 6 interprets the lattice and extracts complex (N-to-1) substitutes.

In the following subsections, we detail each transformation step using our illustrative example.

Operation clustering

In this step, the component groups operations in a set of clusters using Algorithm 4. This algorithm analyzes the similarity matrix for constructing clusters of similar operations. The operations are grouped in the same cluster if their similarity values are greater than or equal to a given similarity threshold (Lines 15-16 of Algorithm 4). The threshold is fixed experimentally. It represents the lower accepted limit for the similarity value to consider two compared objects (messages), in the matrix, as similar.

On the one hand, the identification of clusters reduces the complexity of the computation in the next steps. On the other hand, it reveals simple (1-to-1)

$$Cluster_1\ (\boldsymbol{IP}) = \{IP_{02}, IP_{07}, IP_{15}\}$$

$$Cluster_2(\boldsymbol{W}) = \{W_2, W_4, W_6\}$$

$$Cluster_3(\boldsymbol{Z23}) = \{Z_{23}\}$$

Figure 6.6: Clusters identified from SimMat

	W15.I	W15.O	IP.I	IP.O	W.I	W.O	Z23.I	Z23.O
W15.I	1		0,935	0,579	0,381	0,577	0,491	0,362
W15.O		1	0,108	0,423	0,611	0,684	0,254	0,119
IP.I	0,935	0,108	1		0,362	0,095	0,119	0,502
IP.O	0,579	0,423		1	0,095	0,121	0,901	0,124
W.I	0,381	0,611	0,362	0,095	1		0,547	0,789
W.O	0,577	0,684	0,095	0,121		1	0,535	0,483
Z23.I	0,491	0,254	0,119	0,901	0,547	0,535	1	
Z23.O	0,362	0,119	0,502	0,124	0,789	0,483		1

Figure 6.7: Clusters similarity matrix (*CLSimMat*)

substitutes; i.e. operations in the same cluster which are simple substitutes for each other.

For instance, if we consider the operation clustering for our example using this algorithm, the similarity matrix (depicted in Figure 6.5), with a similarity threshold θ' that is equal to 0.65 (θ'=0.65), produces the 3 clusters (IP, W, and Z23) shown in Figure 6.6.

Afterwards, the similarity matrix has to be reconstructed based on the identified clusters as we will explain in the following subsection.

Similarity matrix reconstruction

The component reconstructs the original similarity matrix (*SimMat*) based on the identification clusters. The new generated matrix is the Cluster Similarity Matrix (*CLSimMat*). The component reduces the number of lines and rows in the original matrix (*SimMat*). Thus, it keeps one operation from each cluster, and it removes the remaining operations. Note that each operation is represented by two lines (two columns) in the matrix. The first line (column) represents its input message and the second its output message.

For instance, Figure 6.7 shows the cluster similarity matrix (*CLSimMat*) which resulted from the reconstruction of the similarity matrix *SimMat* depicted

Algorithm 4 Operation_Clustering

Input: *OP, SimMatrix, similarity Threshold θ'*
Output: *Clusters*
Begin

```
1: Boolean Added ;
2: int nbc = 0 ;
3: while (OP!=φ) do
4:     op = getElement(OP);
5:     Added = false ;
6:     DeleteElement op from OP ;
7:     if (nbc==0) then
8:         nbc ++ ;
9:         Create cluster_nbc ;
10:        Add op To cluster_nbc ;
11:    else
12:        int i ; Added = false;
13:        while (i ≤ nbc) do
14:            op' = getElement(cluster_i) ;
15:            if (GetSimScore(SimMatrix, op.InputMessage,
       op'.InputMessage)≥ θ' && GetSimScore(SimMatrix, op.OutputMessage,
       op'.OutputMessage)≥ θ') then
16:                Add op To cluster_i;
17:                Added = true;
18:                break;
19:            else
20:                i++ ;
21:            end if
22:            if (!Added) then
23:                nbc++;
24:                Create Cluster_nbc;
25:                Add op To Cluster_nbc;
26:            end if
27:        end while
28:    end if
29: end while
```

End

	W15.I	W15.O	IP.I	IP.O	W.I	W.O	Z23.I	Z23.O
W15.I	1		0,935					
W15.O		1				0,684		
IP.I	0,935		1					
IP.O				1			0,901	
W.I					1			0,789
W.O		0,684				1		
Z23.I				0,901			1	
Z23.O					0,789			1

(a) The scaled Matrix

	W15.I	W15.O	IP.I	IP.O	W.I	W.O	Z23.I	Z23.O
W15.I	x		x					
W15.O		x				x		
IP.I	x		x					
IP.O				x			x	
W.I					x			x
W.O		x				x		
Z23.I				x			x	
Z23.O					x			x

(b) The Context matrix (*ContextMat*)

Figure 6.8: Scaled and context matrix

in Figure 6.5, based on the identified clusters presented in Figure 6.6.

Scaling and context matrix building

In this step, the component scales the matrix according to a given similarity threshold θ'. The component removes, from the cluster similarity matrix (*CLSimMatrix*), all the values that are less or equal to the threshold θ'. The removed values are presented by blank cells in the similarity matrix (e.g., matrix (a) in Figure 6.8). The deletion of these values represents the elimination of all relationships between the messages that are not considered as similar. Thus, they will not appear in the context matrix nor in the generated lattice later.

Then, the component replaces all the remaining values in the matrix which are replaced by 'x' to obtain the context matrix (*ContextMat*). For illustration, Figure 6.8 shows in (a) the scaled version of the *CLSimMatrix* presented in Figure 6.7, and in (b) its context version based on the similarity threshold θ' that is equal to 0.65.

We recall that both objects and attributes of the built context (presented by *ContextMat*) are input and output messages, and the relationship between objects and attributes is the similarity relationship.

Square concepts extraction

In this step, the component analyses the context matrix (*ContextMat*) to extract groups of messages that have mutual similarity relationships. These groups of messages are used by the component for detecting composability relationships between operations. These groups are presented in the context matrix by square concepts. By definition, a square concept is a collection of objects with equal extension and intention sets. They are better viewed in the context matrix

	W15.I	IP.I	W15.O	W.O	IP.O	Z23.I	W.I	Z23.O
W15.I	x	x						
IP.I	x	x						
W15.O			x	x				
W.O			x	x				
IP.O					x	x		
Z23.I					x	x		
W.I							x	x
Z23.O							x	x

(a) The Interchanged Context Matrix (*IContextMat*)

G1 = $\{W_{15}.I, IP.I\}$
G2 = $\{W_{15}.O, W.O\}$
G3 = $\{IP.O, Z_{23}.I\}$
G4 = $\{W.I, Z_{23}.I\}$

(b) The Identified groups

Figure 6.9: Square Concepts, Interchanged Context Matrix and identified groups

	{W15.I , IP.I}	{W15.O , W.O}	{IP.O , Z23.I}	{W.I , Z23.O}
W15	x	x		
IP	x		x	
W		x		x
Z23			x	x

Figure 6.10: Operation-group context matrix (OPGContextMat)

by interchanging lines and columns. For instance, Figure 6.9 shows the square concepts marked by the blue squares in the interchanged context matrix (a). In addition, the component forms four corresponding groups as it is depicted in part (b) of the figure.

Operation-Group Context building

According to the groups identified in the previous step, the component builds a new context matrix called the Operation-Group Context Matrix (*OPGContextMat*). The objects of the new context are the operations. The attributes are the groups of mutually similar messages (objects of the square concepts) identified previously. The relationships between the objects and the attributes are the membership of at least one message in the attribute to one operation. Figure 6.10 shows the *OPGContextMat* constructed for our example based on the groups identified in part (b) of Figure 6.9.

Iteration 0 : (Initial groups)
G1={W_{15}.I , IP.I} G2={ W_{15}.O, W.O} G3={IP.O, Z_{23}.I} G4={W.I, Z_{23}.O}
Iteration 1 :
G1'={W_{15}.I-***W_{15}*.O** , *IP*.I- ***IP*.O** } **G2'**={ W_{15}.O, *W*.O} **G3'**={*IP*.O, Z_{23}.I- **Z_{23}.O** } **G4'**={W.I - **W.O** , Z_{23}.O}
Iteration 2 :
G1'={W_{15}.I-***W_{15}*.O** , *IP*.I- ***IP*.O=>Z_{23}.I** } **G2'**={ W_{15}.O, *W*.O} **G3'**={*IP*.O, Z_{23}.I- **Z_{23}.O => W.I** } **G4'**={W.I - **W.O** , Z_{23}.O}

.....

Iteration 4 :
G1'={W15.I - **W15.O** , IP.I - **IP.O => Z23.I - Z23.O =>W.I - W.O**} **G2'**{W15.O , W.O} **G3'** {IP.O , Z23.I - **Z23.O =>W.I - W.O**} **G4'**= {W.I - **W.O** , Z23.O}

Figure 6.11: Extended groups

Group Extending

In this step, the component extends the attributes of the *OPGContextMat* to hold composition sequences. These consequences show the composition relationships between the operations (e.g., operation 1 is composed with operation 2 and the latter is composed with operation 4 and so on). These consequences are built based on the similarity between messages identified from the square concepts identified in Section 6.4.5.

The component extends the attributes (extension of the *OPGContextMat*) by applying the following steps:

- **Step 1**: for every operation input, add its output (e.g. $Op_1.In$ becomes $Op_1.In - Op_1.Out$).
- **Step 2**: for every operation output, add its similar inputs by looking at the initial groups (e.g. $Op_5.Out$ becomes $Op_5.Out \Rightarrow Op_1.Out$).
- **Step 3**: repeat step 1 and step 2 until no change occurs.

The application of these steps on the groups, depicted in part (b) of Figure 6.9, is illustrated in Figure 6.11. The component produces 4 extended groups

	{W15.I-W15.O , IP.I-IP.O => Z23.I-Z23.O =>W.I-W.O}	{W15.O , W.O}	{IP.O , Z23.I-Z23.O => W.I-W.O}	{W.I - W.O , Z23.O}
W15	x	x		
IP	x		x	
W	x	x	x	x
Z23	x		x	x

Figure 6.12: Final Concept Matrix

(shown in Figure 6.11). These groups contain different composition sequences such as "Z23.I- Z23.O => W.I-W.O" in group G3'.

Final Context Building

In this step, the Context Builder and the FCA Classifier component uses the extended groups as attributes for building the final context matrix (*FinContMat*). The objects of *FinContMat* are the same operation candidates. Crosses are added in this context if one operation has at least one message in the corresponding attribute.

For instance, Figure 6.12 shows the final context of the illustrative example.

FCA classification and lattice generation

In the final transformation step, Component 5 uses Concept Explorer ([190]) to generate the operation lattice corresponding to the final formal context elaborated previously. The lattice classifies operations in related groups. Each group has common attributes (intention). In this lattice, the attributes contain the possible substitutes for each operation. Component 6 interprets the lattice and extracts complex (N-to-1) substitutes for the failed operation in the defected web service. Figure 6.13 depicts the operation lattice associated to the final context shown in Figure 6.12.

6.4.6 Lattice interpreter

The Lattice Interpreter (Component 6 in Figure 6.1) queries the generated lattice for determining the substitutes of the failed operation. More precisely, the component parses the intent label of the failed operation to extract composition sequences that could be appropriate substitutes. The component uses the following steps for the interpretation of the intent label:

- **Step 1**: operation input and its output separated by the minus sign (-) are replaced by the name of the operation itself (e.g. $Op_1.In$-$Op_1.Out$ becomes Op_1.)

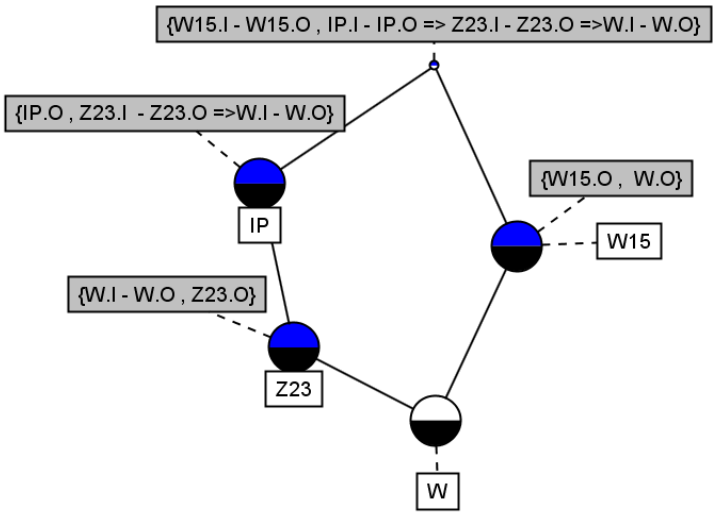

Figure 6.13: Final Lattice

- **Step 2**: the sign (⇒) between operations means composability between them (e.g. $Op_1 \Rightarrow Op_2$ means Op_1 can be composed with Op_2).
- **Step 3**: if the intent holds an orchestration (⇒ sign) and all the elements in that orchestration are operations (but not only the input or the output of operation as $Op.In$ or $Op.Out$), then the sequence of operations (orchestration) is a potential substitute (N-to-1 substitute) for the selected object (failed operation).

For our example, if we select the object labeled $W15$ in the lattice (the failed operation), and we look at its intent (see Figure 6.14), we obtain the following:

- **W15.O, W.O**: means that the output message of operation $W15$ is similar (equivalent) to the output message of operations in $cluster_2$(W). This does not offer any valuable information.
- **W15.I-W15.O, IP.I-IP.O⇒Z23.I-Z23.O⇒W.I-W.O**: if we follow step 1 we obtain **W15, IP⇒Z23⇒W**. The first part (parts are separated by semicolons) does not offer any valuable information because operation **W15** is similar to itself. For the second part, if we follow steps 2 and 3 we get an orchestration of three operations **IP⇒Z23⇒W**. This orchestration is a substitute for the failed operation **W15**.

Additionally, IP and W represent $Cluster_1$ and $Cluster_2$. So, elements in the same cluster can be used interchangeably. Consequently, we obtain a set of substitutes (6 orchestrations), as illustrated in Figure 6.15.

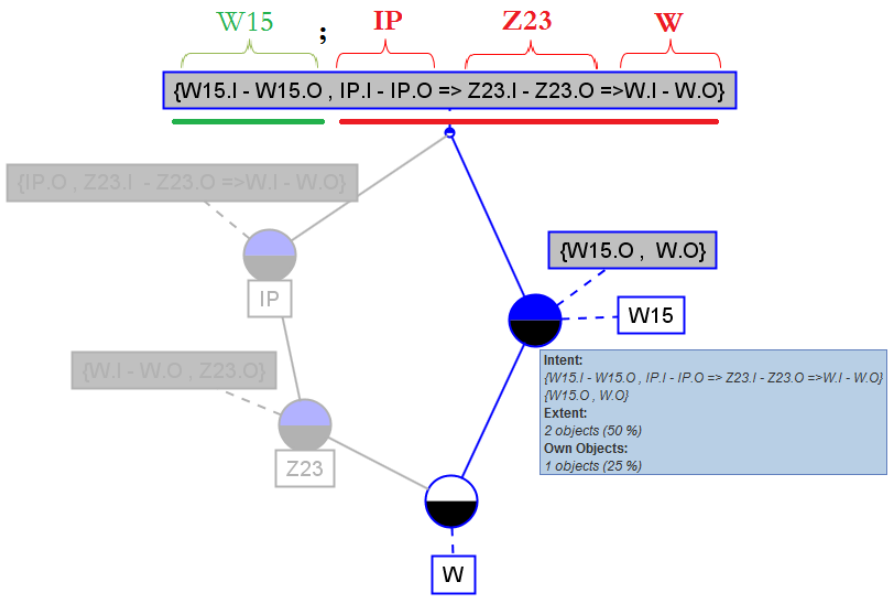

Figure 6.14: Final lattice interpretation

The lattice interpreter component returns a set of substitutes for the failed operation. These substitutes are simple operations (1-to-1 substitutes), and sets of operations that form composite sequences which are considered as complex (N-to-1) substitutes.

6.4.7 Reliability Analyzer

Now that we have a set of relevant substitutes, the reliability analyzer selects among the identified substitutes those who satisfy the minimum requirement of QoS attributes. That is, the services that has greater of equal QoS value than the QoS value of the failed service.

Let S_f be the failed service, and $Q_{S_f} =< q_{f,1}, q_{f,2}, .., q_{f,k} >$ be its associated QoS vector. Note that vectors values could be the minimum QoS values requested in the Service Level Agreement.

Let $S_i, i = 1, .., m$ be the identified substitutes (simple and/or complex), where a complex Substitute $S_j = S_{j,1}, S_{j,2}, .., S_{j,t}$ where t is the number of elementary services in service S_j. Each service S_i has a QoS vector Q_{S_i} defined as follows:

$$Q_{S_i} =< q_{i,1}, q_{i,2}, ..., q_{i,k} >, \quad i = 1, 2, ...m$$

where, k is the number of the QoS parameters, and m is the number of sub-

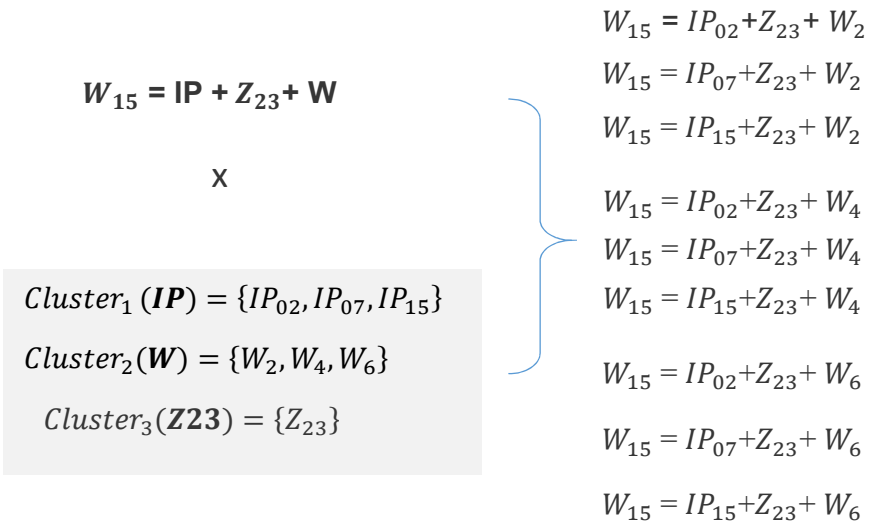

Figure 6.15: List of (N-to-1) substitutes

stitutes. In case of complex substitutes, each $q_{i,j}$, $j = 1, ..k$ is evaluated as follows:

$$q_{i,j} = f(q_{i,j,S_1}, q_{i,j,S_2}, .., q_{i,j,S_t})$$

Where, f is the QoS aggregation function. It calculates the QoS of a composition from the QoS values of its elementary services.

Once all QoS vectors are generated, the services are selected based on the following formula :

$$\max(\Pi(S_1, S_f), \Pi(S_2, S_f), .., \Pi(S_m, S_f))$$

Where, the utility function Π is evaluated as follows :

$$\Pi(S_j, S_f) = \sqrt{(\frac{q_{j,1}}{q_{f,1}})^2 + (\frac{q_{j,2}}{q_{f,2}})^2 + ... + (\frac{q_{j,k}}{q_{f,k}})^2}$$

The max $\Pi(S_j, S_f)$ indicates that service S_j is the most reliables substitutes for $S_f$among the identified substitutes.

6.5 Experiment and validation

According to the previous descriptions, the key concept of the approach is the similarity between a web service orchestration and single web service candidates (similarity between the input/output of the orchestration with the input/output

of the substitute). So, starting from a given similarity degree (threshold), we consider the service as equivalent/substitute to the orchestration. Thus, if any of these substitutes fails, it effortlessly could be replaced by its equivalent orchestration (1-to-N substitution) and vice-versa (N-to-1 substitution). We performed some experiments on a set of real web services to evaluate the approach efficiency and effectiveness. The experiments are conducted following four steps: data set selection, orchestration extraction, substitute extraction, and finally experiment evaluation.

6.5.1 Methodology

We have conducted the experimental process on following these steps:

- Select a set of web services (dataset),
- Extract possible combinations that could be considered as service orchestrations,
- Manually check the obtained combinations to define the set of orchestrations,
- Extract possible substitutes for each orchestration,
- Manually check the obtained substitutes,
- Measure results in terms of well know metrics (recall, precision, accuracy and F1-score).

6.5.2 Data selection

In this experiment, we have selected 64 web service WSDL documents from the WS-Dream dataset ([197]. In fact, the data set hold 3792 web services, but we limited the number of used services to 64 in order to be able to check manually the obtained results in reasonable time. The first service is a weather service, and the remaining services are selected among the others using the selection and filtration algorithms presented previously. The list of used services and experiment results are accessible for download at the following address: https://sites.google.com/site/wservicesubstitues/

6.5.3 Orchestration extraction

From the dataset, we extracted all possible orchestrations from the web services. We assessed the similarity values between these web services, more precisely

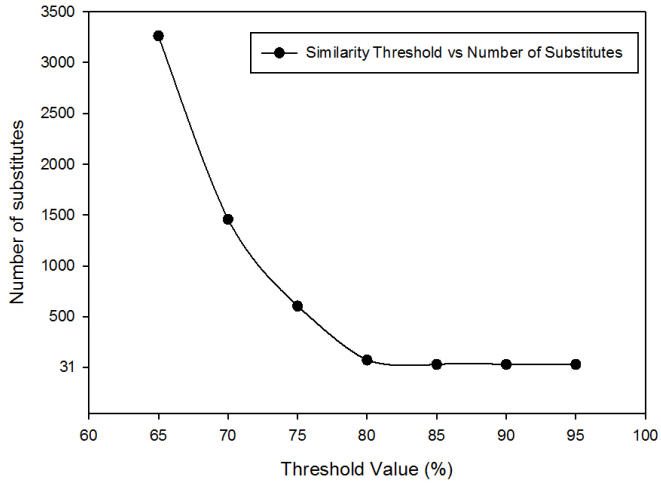

Figure 6.16: Number of obtained substitutes

between their operations. We fixed the composability threshold (θ) at 0.95; which means that two operations are considered composable if the similarity between the output of the first operation and the input of the second operation is greater or equal to 0.95. We selected this score after several experiments to obtain a reduced number of orchestrations. Using this composability threshold (θ= 0.95) we obtained 113 orchestrations. The manual verification of these extracted orchestrations validated the composability between these combinations. Lower thresholds allowed the selection of a larger number of combinations. But a manual adaptation between inputs and outputs is needed to consider these combinations as acceptable.

6.5.4 Substitute extraction

In the next set of experiments, we used different similarity thresholds to automatically extract possible substitutes for each element in the orchestration set. Figure 6.16 shows the values of the extracted substitutes according to different similarity threshold values (θ'). Obviously, we obtain a larger selection once we use relaxed similarity threshold values ($\theta' \leq 0.70$ and $\theta' \geq 0.60$). Nevertheless, the manual examination shows that relaxed threshold values leads to obtain a lot of substitutes that need a manual adaptation (parameters of complex type to adapt). Consequently, we fixed the threshold value (θ') to 0.75.

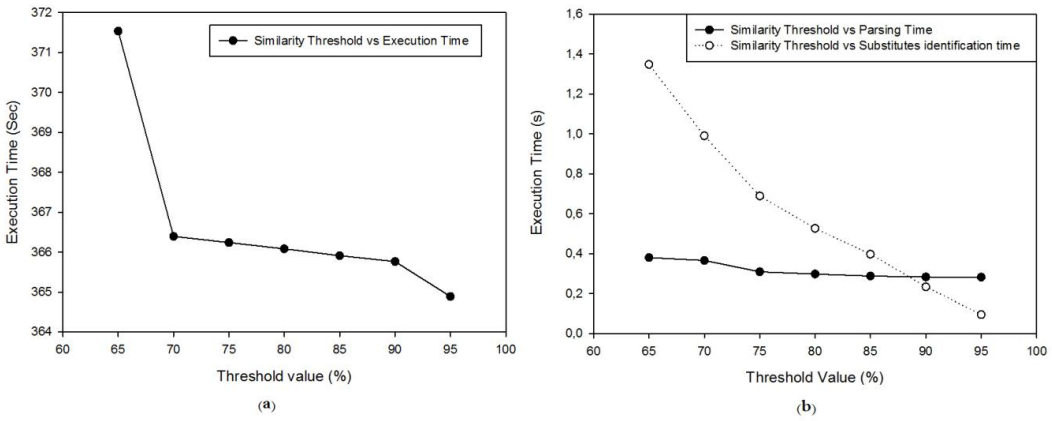

Figure 6.17: Experiments Execution time

6.5.5 System performances

The machine used for this experiment is running with an Intel processor (I3-2100 CPU 3.10 GHZ) and RAM of 4 GB under Windows 7 Operating System (64 bits). Figure 6.17 summarizes the execution times for each run depending on the chosen similarity threshold value. Execution times in part (a) of Figure 6.17 include execution times for WSDL parsing, similarity measurement between services, similarity matrix reconstruction, combination of operation extraction, and finally matrix analysis and substitute identification. Part (b) of the same figure depicts the obtained values for the WSDL parsing and substitute extraction times.

The WSDL document parsing time ranges from 0.3 to 0.4 seconds for the same number of WSDL files. The analysis of the similarity matrix to identify substitutes ranges from 0.07 to 1.4 seconds. These values depend on the matrix size which is correlated with the number of operations selected based on the used similarity threshold. The values depicted in part (a) of Figure 6.17 are the execution times of the experiment runs. In each run, the WSDL documents are parsed, the possible combinations (orchestration) are extracted, the similarity matrix is constructed and then it is analyzed and the substitutes are identified based on the fixed similarity threshold. These execution times range from 360 to 372 seconds. They are inversely proportional with the similarity threshold, thus, with the number of selected operations.

	Number of orchestrations	Number of substitutes	
		By Tool	Manually checked
Θ=0.95 Θ'=0.75	113	605	442
Θ=0.95 Θ'=0.73		645	452

(a)

TP (True Positive)	FP (False Positive)
442	163
FN (False negative)	TN (True Negative)
10	30

(b)

Figure 6.18: Result of manual verification of extracted substitutes

6.5.6 Result measurement

Finally, we measure the effectiveness of the approach by calculating some information retrieval metrics such as recall, accuracy and precision. These metrics have been broadly used in the context of web service discovery and selection ([68], [139]).

In Figure 6.18, table (a) shows the number of extracted substitutes according to the fixed thresholds. Column "By tool" depicts the number of the substitutes that are extracted automatically; that are considered by the tool as correct substitutes for the orchestrations. Column "Manually checked" contains the number of substitutes that are manually verified. These values are used to identify:

- The number of true positives (number of substitutes that are identified by the tool and that are correct),
- The number of the false positives (number of substitutes that are identified by the tool and that are incorrect),
- The number of false negatives (number of substitutes that are correct but that are not identified by the tool),
- The number of true negatives (number of substitutes that are incorrect and that are not identified by the tool).

Values summarized in table (b) Figure 6.18 are used to calculate the following IR metrics:

- **Precision**: is the number of true results divided by the number of all returned results.

  Precision $(P) = \frac{TP}{TP+FP} =$ **0.731**

- **Accuracy**: is the number of true results (both true positives and true negatives) in the obtained results.

  Accuracy $(A) = \frac{TP+TN}{TP+FP+FN+TN} =$ **0.732**

- **Recall**: is the number of correct results divided by the number of results that should be returned.

  Recall $(R) = \frac{TP}{TP+FN} =$ **0.98**.

- **F1-Score**: is a measure of test accuracy. It can be interpreted as a weighted average of both precision and recall.

  F1-Score= $2 \times \frac{P \times R}{P+R} =$ **0.836**.

The experiment shows that the precision rate is relatively low because of the number of the false positives. These false positives are the operations that are identified by the tool and considered as substitutes for some orchestrations. The manual verification shows that these operations are semantically different than the orchestration, but the tool selected them as substitutes because their input or output messages are syntactically similar to the input of the first operation in the orchestration or, respectively, the output of the last operation in the orchestration. However, the experiment shows that the accuracy rate is very high, and this is very important in the case of identification of substitutes.

In summary, the metric values obtained from this experiment on the one hand, and the case example that illustrated the section presenting the approach on the other hand, both show the practicability of the approach. Therefore, we can effectively retrieve (1-to-1) and (N-to-1) substitutes for failed services. Nevertheless, this experiment showed some drawbacks that we could avoid by taking into account the following:

- The better we measure the similarity and we fix its threshold the better results we obtain.
- Relaxed thresholds have to be avoided for substitutes identification.
- In case we choose medium thresholds, some substitutes need an adaptation (casting, complex type conversion, etc.) to replace failed operations. These service adaptations can be addressed at protocol level [115].

6.5.7 Threats to validity

Conclusion, construct, internal and external threats are the four categories of threats to validity proposed in [185]. An extension of this framework to cover Search-Based Software Engineering (SBSE) experiments is proposed by Barros and Dias-Neto [17]. In this section, we discuss the threats to validity of the conducted experiment according to the former extension.

Conclusion validity threats have a concern with the relationship between the treatment and the outcome. The empirical design must make sure that there is a statistical relationship between the involved parts. The main conclusion threats include the non-consideration of random variation, the lack of good descriptive statistics and the luck of the use of a meaningful baseline [17]. In this experiment, we addressed the first threat by having many runs for each fixed threshold to measure the execution time. We note here that the number of experiment runs did not affect the number of obtained substitutes. Moreover, we cope with the remaining conclusion threats by comparing the obtained results with the manually identified ones, which is considered as a solid comparison baseline.

Construct validity threats are concerned with the relation between theory and observation, ensuring that the treatment reflects the construct of the cause and that the outcome reflects the construct of the effect. In SBSE experiments, construct threats involve using invalid efficiency and effectiveness measures and not discussing the underlying model subject to optimization [17]. We coped with these threats by discussing the cost measures of the experiments (the execution time). In addition, we addressed the validity of the effectiveness measures using the recall, precision and accuracy metrics, which are widely used in such experiments.

Internal validity threats are concerned with the evaluation of the causality of the relationship between the treatment and the outcome in an experimental study, or the result of a factor upon which the researcher has no control. Internal threats may include: 1) poor parameter settings, 2) luck of discussion on code instrumentation, 3) luck of clear data collection procedures, and finally 4) the luck of a real problem instance [17]. In this experiment, we cope with these threats by presenting the most important parameter (similarity and composability thresholds) used in the experiment, by providing the source code of the similarity assessor used in the approach, by conducting the experiment using a set of real web services, and finally by describing the data collection procedure.

External validity threats are concerned with theg generalization of the observed results to a larger population, outside the sample instances used in the

experiment. Specifically, these threats include the lack of a clear definition of target instances, the lack of clear instance selection strategy, and the fact of not having enough diversity in instance size and complexity [17]. In this experiment, the external threats come from the limited number of instances of selected web services used in the experiment. Even if we have selected these services from a large collection of (almost 3800) real web services, it is still difficult to generalize these results because they depend on the studied set of web services. Nevertheless, the goal of this experiment is to show the practicability and efficiency of the proposed substitute identification process on real-world data.

6.6 Summary

In this chapter, we proposed an approach for identifying relevant web service substitutes. The approach relies on measuring the similarity between web service interfaces. We presented the necessary algorithms and techniques for selecting, filtering, and clustering web service candidates. We used a similarity matrix to determine the relation between services. We described the steps that aim to find simple substitutes (1-to-1) and complex (N-to-1). We incorporated FCA to classify and visualize the relevant results. We have shown the practicability of the approach using a case example. In addition, we validated the approach via a set of experiments conducted on a collection of real web services.

In the next chapter, we propose a framework for evaluating reputation scores of web services and their providers. Therefore, reputation scores could be employed during reliability analysis and verification.

Chapter 7

Web Service Reputation Management Framework

Contents

7.1 Overview

Reputation of web services and trustworthiness of its provider is important criterion when selecting reliable atomic services. Because reputation indicates how the service is perceived from users after consummation and experimentation. Evaluating reputation is challenging issue due to the points presented in (section intro). In this chapter, we introduce a reputation management framework for web service and web service orchestration. This solution provides some solutions to the deficiencies and limitations discussed in (Section).

7.2 Reputation management framework

In this section, we present first the reputation management framework for trustful web service recommendation. Then, we describe the main components and their roles for acquiring, storing, and aggregating user feedback ratings.

7.2.1 Framework architecture

Figure 7.1 depicts the architecture of the reputation Management Framework. The framework permits to users the search for web services by providing search queries or by direct browsing of registries via *Search and Selection Interface* (Component 1). After service consumption, the user could send a feedback about her/his satisfaction to the system via the *Feedback Collector* (Component 2). Collected feedback ratings are then stored in the feedback database.

The *Reputation Manager* (component 3) reassesses periodically the reputation of the web services based on new modifications in the feedback database, using the assessment model presented in Section 7.3. Component 3 updates the Reputation Database by new assessed values.

The Search and Selection Interface recommends for users sorted sets of web services that correspond to the provided search queries. These services are processed by the *Service Recommender* (Component 4) which: 1) retrieves services

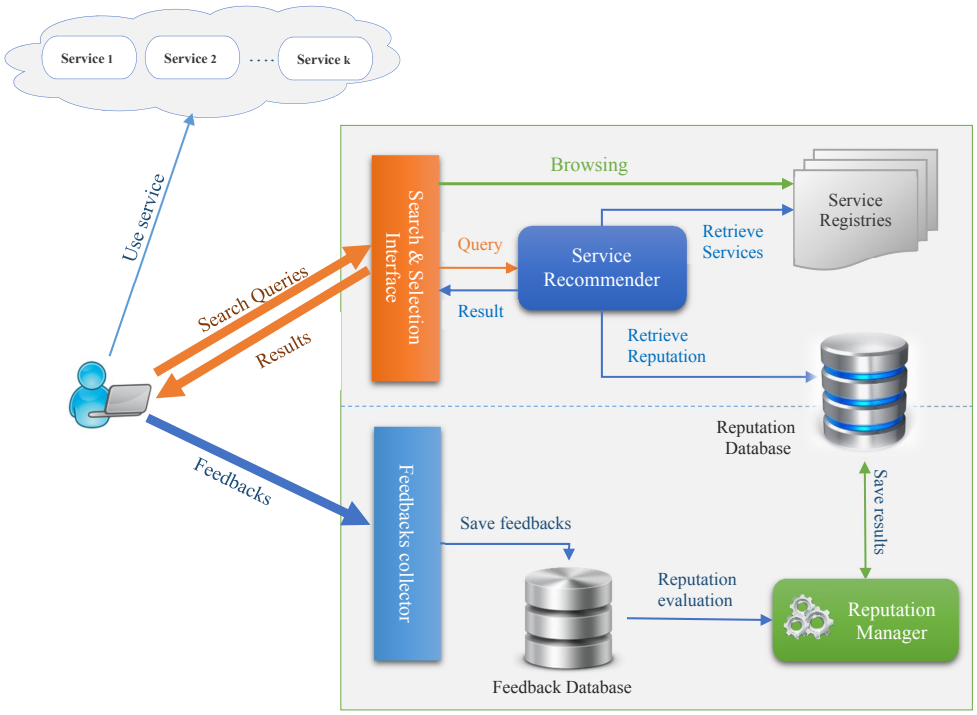

Figure 7.1: Architecture of the Reputation Manager

from registries, 2) extracts reputation scores from the reputation database, 3) sorts services based on their reputation scores, and 4) delivers results to Component 1 which recommends them to users.

7.2.2 Feedback collector

The role of the Feedback Collector is to provide a human-interface for service users that enables them to submit their feedbacks. User feedback is a quantification of user's opinion about the consumed service. In the proposed architecture, feedbacks are user ratings that range in a scale of 10, where 0 represents a complete dissatisfaction and 10 a total satisfaction. Every web service has a unique identifier. Therefore, during feedback submission the user has to provide the service ID and the attributed feedback rating. During each feedback transaction, the component stores in the feedback database the following information:

- Feedback_ID: represents the identifier of the current feedback record.
- User_ID: represents the user identifier. In order to avoid user subscription. The system considers the IP address of the user as its identifier.
- Service_ID: represents the identifier of the consumed web service.

- Rating: represents the rate assigned by the user to the consumed service. As mentioned previously, rates are unsigned integers that range between 0 and 10.
- Timestamp: represents the time of feedback reception.
- Modification_Nbr: represents the number of update of the record. Initially it takes the value 1.

Due to performance issues, for the same service and the same user, the feedback collector stores in the database only one record during an amount of time T' (for example T'=24 hours). This record is updated each time a new rating is introduced (feedback from the same user for the same service) within that time interval. The feedback collector ignores additional updates on the same record once the number of modifications reaches N (for example N=5).

7.2.3 Reputation manager

The Reputation Manager reassesses the reputation of web services by aggregating feedbacks stored in the feedback database. We can summarize the functionality of this component as follows:

- It retrieves new feedback records since the last assessment round from the feedback database.
- It selects, from the retrieved records, the services that their reputations have to be assessed or updated.
- It reevaluates the credibility (honesty factor) of the raters in respect to the model proposed in subsection 7.3.4
- It extracts all past ratings of each selected service, and assesses its reputation following the assessment model presented in Section 7.3.
- The assessment results are stored in the reputation database. The stored records have the following structure:
  1. Service_ID: represents the identifier of the web service in the system.
  2. Reputation: represents the assessed reputation value for the service.
  3. Timestamp: represents the time of the last assessment round.
- The reputation manager starts new assessment round every time slot T. T by default is 24 hours, but it could be fixed in regard to system performances.

7.2.4 Search and selection interface

The Search and Selection interface enables users to interact with the system for selecting web services. The user via this component can browse directly the services registries. Moreover, it can handle search queries. These queries are generally a set of keywords describing the sought services. The interface transfers the query to the service selector component which analyzes the query and searches for the appropriate services that match the query. The selector return a sorted list of web services to the interface, which by its turn, returns it as result to users.

7.2.5 Service recommender

The service recommender component processes user queries through the following three steps:

- **Step 1: Query preparation**: the component analyses and prepares the query by removing stop words, stemming the remaining keywords, and adding synonyms to the query. The result of this step is a bag of words that represents the initial search query.

- **Step 2: Service retrieval**: the selector component searches for services that hold at least one element in the bag of words. We suppose that services in the registries are tagged using one of the techniques proposed by Azemh *et al.* [10] and Falleri *et al.* [60]. We assume that similarity values between web services are assessed using the approach presented by Tibermacine *et al.* [165]. This information has to be stored in a database where two relations are defined as follows:

  1. *Services Relation*: this relation holds the keyword of each service. It has three named attributes represented in Table 7.1.
  2. *Similarity Relation*: this relation holds similarity values between web services. It holds the attributes grouped in Table 7.2.

- **Step 3: Reputation-based sorting**: After selecting web services, the component retrieves the reputation value of each service in the result set from the reputation database. Then, the component groups and sorts services based on their reputation and similarity values. Recommended results are sent back to the interface component.

Table 7.1: Attributes of the service relation

Attributes	Description
Service_ID	The identifier of the web service
WSDL_Path	The relative path of the WSDL document of the service
TagList	A list of keywords associated with the service
EntryDate	The date of the entry of the service in the registry

Table 7.2: Attributes of the similarity relation

Attribute	Description
Service1_ID	The identifier of the first web service
Service2_ID	The identifier of the second web service
Similarity_Score	The similarity score between web service 1 and web service 2

7.3 Reputation assessment model

Reputation is assessed by aggregating previous user feedback ratings. In this section, we propose a model for assessing web service reputation scores based on collected feedback ratings. First, we discuss the considered metrics and we provide the reputation assessment formulas. Besides, we present a bootstrapping mechanism for evaluating the reputation of newcomer services. in addition, we show how to evaluate the reputation of service providers and web service orchestration.

7.3.1 Evaluation metrics

We enclose different metrics to evaluate web service reputation by aggregating user feedback ratings. Some of these metrics are already taken in related work such as in [101,108,187]. However, these propositions are using some metrics and neglecting others. We build our reputation assessment model upon the following metrics:

1. **User honesty (credibility)**: The credibility of user has its impact on

provided feedback ratings; a dishonest user can dramatically decrease the reputation of good service or increase the reputation of a poor service. Therefore, it is essential for accurate reputation measurement to consider the level of user credibility during reputation assessment.

2. **User rating history**: Users may behave maliciously; they can start as an honest users, then they change their behavior by time. In consequence, the assessment framework has to estimate and update the honesty level of users in function of their rating history.

3. **Penalization of suspicious users**: We consider a user as suspicious user when its estimated credibility is less than a fixed threshold. The model neutralize feedback ratings of all suspicious users. This mechanism permits to ensure the purity of feedback ratings used during the assessment of web service reputation.

4. **Feedback History**: The reputation assessment model handles all feedback ratings stored in the database.

5. **Temporal sensitivity**: We include this factor (written λ) in the assessment model to address the temporal sensitivity of ratings, where new feedbacks has more impacts on the assessed reputation than older ones. We adopt this metric from [187] and [183].

7.3.2 Assessment formula

Let $\delta_{(i,k)}$ be the feedback rate of user (i) for service (k). The rating values range between 0 and 10, where 0 represents a total dissatisfaction about the functionalities and QoS of the used web service, and 10 represents its total satisfaction. More precisely, let $\delta_{(i,k)} \in \{\delta^{-}_{(i,k)}, \delta^{+}_{(i,k)}\}$, where $\delta^{-}_{(i,k)} \in \{0,1,2,3,4,5\}$ and $\delta^{+}_{(i,k)} \in \{6,7,8,9,10\}$. We assume that $\delta^{-}_{(i,k)}$ is a negative feedback, and $\delta^{+}_{(i,k)}$ is a positive one.

Let $\Phi(S_k)$ be the sum of all rates weighted by the time sensitivity (aging) factor λ and the rater's honesty score H. Mathematically, we define $\Phi(S_k)$ as follows:

$$\Phi(S_k) = (\sum_{i=1}^{n} \delta^{+}_{(i,k)} \times \lambda^{d_i} \times H_i) + (\sum_{j=1}^{m} \delta^{-}_{(j,k)} \times \lambda^{d_j} \times H_j) \tag{7.1}$$

Where:

- $\delta^{+}_{(i,k)}$ is the i-th positive rating, and n is the number of positive rates for service (S_k).

- $\delta^{-}_{(j,k)}$ is the j-th negative rating, and m is the number of negative rates for service (S_k).
- $n + m$ is the number of all rates for service S_k.
- λ is the aging factor, where $0 \leq \lambda \leq 1$.
- d_i is the age of the rate assigned by user i for service S_k in terms of number of days.
- H_i is the honesty score assessed for user i.

In addition, we define the function $\Omega(S_k)$ of web service S_k as the fraction of the difference between positive and negative feedbacks (weighted by their corresponding aging and honesty factors) to the sum of all feedbacks ($\Phi(S_k)$). We write:

$$\Omega(S_k) = \begin{cases} \frac{(\sum_{i=1}^{n} \delta^{+}_{(i,k)} \times \lambda^{d_i} \times H_i) - (\sum_{j=1}^{m} \delta^{-}_{(j,k)} \times \lambda^{d_j} \times H_j)}{\Phi(S_k)} & if \Phi(S_k) \neq 0 \\ -1 & otherwise \end{cases} \tag{7.2}$$

The range (co-domain) of the Ω function is in the interval $[-1, 1]$. If all the rates of service S_k are positive, $\Omega(S_k)$ equates 1. Conversely, if all rates are negative, $\Omega(S_k)$ is equal to -1.

7.3.3 Reputation

The reputation of service S_k ($R(S_k)$) is calculated as the fraction of the mean of feedbacks at the rating scale (10) if all feedbacks are either positive or negative. Otherwise, the reputation is the normalized value of the Ω function at the interval [0,1]. Formally, we write:

$$R(S_k) = \begin{cases} \frac{(\sum_{i=1}^{n} \delta^{+}_{(i,k)} \times \lambda^{d_i} \times H_i)}{10 \times n} & if\ \Omega(S_k) = 1 \\ \frac{(\sum_{j=1}^{m} \delta^{-}_{(j,k)} \times \lambda^{d_j} \times H_j)}{10 \times m} & if\ \Omega(S_k) = -1 \\ \frac{\Omega(S_k)+1}{2} & Otherwise \end{cases} \tag{7.3}$$

7.3.4 Honesty factor

The honesty factor is the probability that the user gives a honest feedback. By default, a new user is assigned the value ($\frac{1}{2}$). This value means that the user is neither honest nor dishonest. So, feedbacks provided from new users do not affect dramatically the reputation of the ranked services. In the next ranking experiences, we calculate the honesty factor for user i based on its previous experiences as follows:

Rated Services	Other's feedbacks		User (i) feedback	Probability
	Positive (+)	Negative (-)		
WS_7	12	3	-	$\frac{4}{16} = \frac{1}{4}$
WS_{12}	1	10	-	$\frac{11}{12}$
WS_{19}	5	1	+	$\frac{6}{7}$
WS_{58}	16	0	+	1
WS_{62}	6	3	-	$\frac{2}{5}$
			H_i	0,66

Figure 7.2: Example of the assessment of the Honesty Factor

$$H_i = \frac{\sum_{s=1}^{t}\left(\frac{\Psi(s)}{\Psi^+(s)+\Psi^-(s)}\right)}{t}$$

$$\text{such as:} \quad \Psi(s) = \begin{cases} \Psi^+(s) & if(\delta_{i,s} = \delta^+_{i,s}) \\ \Psi^-(s) & if(\delta_{i,s} = \delta^-_{i,s}) \end{cases} \tag{7.4}$$

where,

- t: is the number of services rated by user (i)
- $\Psi^+(s)$: denotes the number of positive ratings for service s
- $\Psi^-(s)$: denotes the number of negative ratings for service s

An illustrative example of the calculation of the honesty factor of a user who sends feedback ratings for the services WS_7, WS_{12}, WS_{19}, WS_{58}, WS_{62} is presented in Fig. 7.2. We can observe that this user rated the first service negatively and belongs to the quarter of users that rated negatively this service. In addition, she/he rated negatively the last service like a minority of users ($\frac{2}{5}$). These values lower her/his honesty. However she/he evaluated the other services like a majority of users. For instance, she/he has rated the second service negatively, like 10 other users (among 11). And this increases her/his honesty. The overall honesty factor for this user has been evaluated to 0.66, which means that, for the moment, this user is likely to be honest.

7.3.5 Suspicious user penalization

In this paper, we consider a user with a honesty factor less than a fixed threshold as a suspicious user. Consequently, the effect of feedback ratings provided by this user has to be neutralized. Thus, we penalize suspicious users by setting their honesty factors to zero. In this way, we ensure that web service reputation is assessed from fair feedback ratings. The honesty of a user is subject to change. This depends on her/his future behavior. The model reevaluates user honesty every time the system gathers new feedback ratings from that user. The aging factor reduces the impact of old malicious feedback ratings in case the suspicious user has enhanced her/his behavior and has provided recent new fair ratings.

7.3.6 Provider reputation

The reputation of the provider mainly depends on the quality of its offered services thus on their reputation. Consequently, we assess the reputation of a provider as the mean of the reputations scores of its services. Given the provide Pr_x, let $Services(Pr_x) = S_1, S_2, ..., S_n$ be the set of services provided by Pr_x. The reputation of this provider is assessed as follows :

$$RP(Pr_x) = \begin{cases} \frac{(\Sigma_{i=1}^{n} R(S_i))}{n} & if \quad Services(Pr_x) \neq \phi \\ \frac{1}{2} & Otherwise \end{cases} \tag{7.5}$$

where,

- $R(S_i)$ is the reputation of service S_i that belongs to provider's service set ($S_i \in services(Pr_x)$).
- ϕ denotes the empty set.

The reputation of a new provider, in case of the introduction of new service, is set to $\frac{1}{2}$. This value means that the provider is neither trusted nor untrusted. The reputation of the provider is updated automatically once the reputation of one of its services has been modified.

7.3.7 WS Orchestration reputation assessment

Several existing reputation models focus on the selection of single services and neglect pure WS orchestrations [103]. Ideally, advanced WS selection systems have to allow to their users to select WS orchestrations among different possibilities, based on reputation and service similarity. Therefore, a user can start by selecting a set of services to construct her/his orchestration. The system assesses the reputation of this orchestration based on the reputation of single services.

Table 7.3: Web service candidate information of the hypothetical WS orchestration

Task	Invocation	Web service	Reputation
T1	3	S_{11}	0.89
		S_{12}	0.77
		S_{13}	0.93
T2	1	$S21$	0.65
		S_{22}	0.25
		$S23$	0.82
		$S24$	0.37
T3	2	$S31$	0.72
		S_{32}	0.68

Then, it suggests similar orchestrations with better reputation values based on possible service substitutes (similar services that can replace initial ones in the orchestration).

We complete the proposed model for allowing reputation-based selection of WS orchestrations. Let us suppose that we have a generic orchestration where n web services $\{S_i \mid i \in [1, n]\}$ are involved. Each service has a reputation value assessed and stored separately in the reputation database (RDB). In addition, the user has to provide the number of invocations for each service in the orchestration. In case the number of invocations is missed, the system will consider that each service is invoked once.

Each service S_i is invoked X_i times in the orchestration, and R_i represents the reputation of the single web service S_i.

Accordingly, we propose to assess the reputation of the orchestration Rep (Cs) as follows:

$$Rep(C_S) = \frac{\sum_{i=1}^{n} X_i \times R_i}{\sum_{i=1}^{n} X_i} \tag{7.6}$$

where,

- X_i is the number of invocations of service S_i in the orchestration CS. By default, it takes the value 1.
- R_i is the reputation of the service S_i.

We believe that, the most invoked service in the orchestration is the service that its reputation value influences the most the overall orchestration reputation.

Table 7.4: The first ten recommended web service combinations for the orchestration based on the estimated reputation values

Rank	Recommended Combination	Estimated Reputation
1	$\{S_{13}, S_{23}, S_{31}\}$	0.842
2	$\{S_{13}, S_{23}, S_{32}\}$	0.828
3	$\{S_{11}, S_{23}, S_{31}\}$	0.822
4	$\{S_{13}, S_{21}, S_{31}\}$	0.813
5	$\{S_{11}, S_{23}, S_{32}\}$	0.808
6	$\{S_{13}, S_{21}, S_{32}\}$	0.800
7	$\{S_{11}, S_{21}, S_{31}\}$	0.793
8	$\{S_{11}, S_{21}, S_{32}\}$	0.780
9	$\{S_{13}, S_{24}, S_{31}\}$	0.767
10	$\{S_{12}, S_{23}, S_{31}\}$	0.762

For illustration, let us suppose that we have a hypothetical orchestration (CS) of three tasks T_1,T_2, T_3. Initially, the system has found a set of service candidates for each task: $\{S_{11}, S_{12}, S_{13}\}$ for T_1, $\{S_{21}, S_{22}, S_{23}, S_{24}\}$ for T_2, and $\{S_{31}, S_{32}\}$ for T_3. Table 7.3 shows the number of invocations and the reputation score of each service candidate. The best recommended combination to achieve this orchestration is $S13$,$S23$, and $S31$. These services have the maximal reputation scores. We assess the overall reputation of this orchestration as follows:

$$Rep(Cs) = \frac{(3 \times 0.93) + 0.82 + (2 \times 0.72)}{6} = 0.84.$$

Table 7.4 lists the first ten recommended combinations for maintaining CS orchestration.

7.4 Reputation bootstrapping Model

Rather than assigning a default initial reputation value to newly posted web services, we propose a reputation bootstrapping model that assigns appropriate reputation values depending on their initial QoS. Thereby, the assigned reputation values give more chances for these services to be recommended and selected. Moreover, the bootstrapping model offers a solution for the "cold start" and the "whitewashing" problems. So, from the one hand we avoid fixed initial values, and from the other hand, even if a service leaves the system and comes back later with another identity (Name, URL, Provider Domain, etc.), it will get a

reputation value that approximates to its reputation value before leaving the system.

At the initial phase, when a new Web service S_i arrives to the system, we assume that it comes with a set of initial QoS vector $Q_{S_i}^{init} =< Q_{i,1}, Q_{i,2}, ..., Q_{i,k} >$. These QoS values are provided during publication time by the service provider ($Pr(S_i)$) as advertised QoS. They can be established by the system after a short period of service testing and monitoring (The system can use one of the approaches proposed in this survey [**?**]). Then, to bootstrap the reputation of the new service S_i, the system goes through three phases: i) provider reputation evaluation, which is explained in Section 7.4.1, ii) reputation estimation from similar services, which is described in Section 7.4.2, and iii) reputation estimation from multiple regression models built from QoS and reputation values of long-standing services, which is detailed in Section 7.4.3.

In Algorithm 5, we present the process that covers the three phases to estimate the reputation of the new web services S_i. First, the system checks whether the provider of the service is known by the system (Line 1 in Algorithm 5), that is, the service provider belongs to the list of providers *ProviderList* that have previously published services in the system. In the positive case, the system computes the reputation of this provider, denoted *prReputation*, based on the reputation of its long-standing services (Line 2). We present details on how we calculate the reputation of the provider in Section 7.4.1.

Afterwards, the system seeks for long-standing services that provide similar functionalities to the new web service (Line 3). To evaluate the similarity between web services, we use the approach proposed by Tibermacine *et al.* [165]. If the *simServiceSet*, which denotes the set of similar service, is not empty, the system builds an equation model from the QoS vectors and reputation values of similar services (Line 5 in the Algorithm), as it is detailed in Section 7.4.2. The system estimates from the built model a reputation value, denoted *mReputation*, for the newcomer service based on its initial QoS vector. The maximum value between the provider reputation and the estimated reputation (*mReputation*) is assigned to the newcomer service. The choice of the maximum value (an optimistic strategy) is motivated by the fact that if a provider has a good reputation, it is likely that its new service will have a good reputation too. If the estimated reputation is better, then we give a chance to this service to be selected and evaluated by users, independently of its provider reputation.

Besides, when the provider of the service is also new to the system, we check if it is a whitewashing situation (lines 12-17 in Algorithm 1). The system retrieves all similar long-standing services and compares their similarity scores with the

Algorithm 5 Reputation bootstrapping algorithm

Input: S_i new service
Output: Reputation
Begin

```
if (Provider(S_i) ∈ ProviderList) then
    prReputation = providerReputation(Provider(S_i));
    simServiceSet = getSimilarServices(S_i,ServiceList);
    if (simServiceSet≠ Φ) then
        eModel = buildEquationalModel(simServiceSet);
        mReputation = estimateReputation(Qos(S_i),eModel);
        reputation= Max(prReputation,mReputation);
    else
        reputation= prReputation;
    end if
else
    simServiceSet = getSimilarServices(S_i,ServiceList);
    if (simServiceSet≠ Φ) then
        SimVector = Similarities(S_i, simServiceSet);
        aService =HighestScoreService(SimVector);
        if (Max(simVector)==1 && hasLeft(aService)) then
            reputation =getReputation(aService);
        else
            eModel = buildEquationalModel(simServiceSet);
            reputation = estimateReputation(Qos(S_i),eModel);
        end if
    else
        rModel = BuildRegressionModel(serviceList);
        reputation = estimateReputation(Qos(S_i),rModel);
    end if
end if
```

End

new service (similarity scores range between 0 and 1, where 1 means that services are totally similar and 0 otherwise). If the highest similarity score equates to one, and the similar service has left the system, then, the provider of the new service becomes suspicious, and we assign the (old) reputation of the service that has left to the new service (line 17 in the algorithm). Otherwise, we go to phase two, where the system builds an equation model from QoS and reputation values of similar services. The estimated reputation is assigned to the newcomer service.

When the newcomer service and its provider are both new, and there are no similar services in the system, we go to phase three (detailed in Section 7.4.3). The system builds a multiple linear regression model from QoS vectors and reputation values of all long-standing web services in the system (line 23 in the algorithm). Likewise to phase two, the model gives also an estimation of service reputation based on the service initial QoS. The estimated value is assigned to the newcomer web service.

7.4.1 Provider reputation-based estimation

The reputation of a given provider mainly depends on the quality of its offered services, thus on their reputation. In this phase, we assess the reputation of a provider as the average of the reputation scores of its services. Given a provide Pr_x, let $Services(Pr_x) = S_1, S_2, ..., S_n$ be the set of web services provided by Pr_x. The reputation of this provider is assessed as follows:

$$RP(Pr_x) = \begin{cases} \frac{(\Sigma_{i=1}^{n} R(S_i))}{n} & if \quad Services(Pr_x) \neq \phi \\ 0 & Otherwise \end{cases} \tag{7.7}$$

where,

- $R(S_i)$ is the reputation of service S_i that belongs to the provider's service set ($S_i \in services(Pr_x)$).
- ϕ denotes the empty set.

The reputation of a new provider, in case of the introduction of a new service, is set to 0. This value means that the provider is by default not trusted. However the reputation of the provider is updated automatically once the reputation of one of its services has been modified.

7.4.2 Reputation estimation from similar services

Since a user rates similar web services based on the same criteria, it could be possible to estimate the reputation of newcomer web service based on reputation

Service	Response Time (T)	Availability (A)	Price (P)	Reputation (R)
S_1	T_1	A_1	P_1	R_1
S_2	T_2	A_2	P_2	R_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
S_m	T_m	A_m	P_m	R_m

Table 7.5: Generic form of service QoS vectors and reputation

values of its similar services, which are the aggregations of user feedback ratings. In this phase, we present how to estimate, according to QoS values, the reputation of a newcomer service S_i. For illustration, we suppose that service S_i comes with three initial QoS values: i) response time, denoted T_i or in general form $Q_{i,1}$; ii) availability, denoted A_i or $Q_{i,2}$; and iii) price, denoted P_i or $Q_{i,3}$.

To estimate the reputation of service S_i, we follow the next steps:

1. *Selecting similar services:* First, the system selects from its database long-standing services that are similar to service S_i. We use the approach proposed by Tibermacine *et al.* [165] to assess the similarity between the new web service and the long-standing services. The approach assesses the similarity between two services by comparing their WSDL files using several lexical and semantic metrics. The results of the similarity assessment are scores that range between 0 and 1, where 0 represents a total dissimilarity and 1 a total similarity. Compared web services with a similarity score greater or equal than a fixed threshold (e.g., 0.75) are considered as similar. The result of this step is a set of similar services denoted by $simServiceSet$.

2. *Preparing QoS data:* Second, the system retrieves QoS and reputation values of each service in $simServiceSet$. Let $S_j \in simServiceSet$ (j=1,..,m) be a similar service. Each similar service S_j has a QoS vector $Q_{S_j} =< T_j, A_j, P_j, R_j >$. Where R_j denotes the reputation of service S_j calculated from user feedback ratings. Table 7.5 groups the collected data. Besides, the newcomer service S_i has the vector $Q_{S_i}^{init} =< T_i, A_i, P_i, \hat{R}_i >$. Where T_i, A_i and P_i represent the initial QoS values, and $\hat{R}_i$ is the unknown reputation value.

   Afterwards, the system scales all QoS values in the interval $[0, 1]$. Thus, each QoS value ($QosVal$),which is T_j, A_j, or P_j (j=1,..,m) in Table 7.5, is replaced by $NewQosVal$ as follows :

$$NewQosVal = \frac{QosVal - MinVal}{MaxVal - MinVal} \tag{7.8}$$

	S_τ	S_i
Response Time (T)	$T_\tau = \frac{\Sigma_{i=1}^{m} T_i}{m}$	T_i
Availability (A)	$A_\tau = \frac{\Sigma_{i=1}^{m} A_i}{m}$	A_i
Price (P)	$P_\tau = \frac{\Sigma_{i=1}^{m} P_i}{m}$	P_i
Reputation (R)	$R_\tau = \frac{\Sigma_{i=1}^{m} R_i}{m}$	$R_i?$

Table 7.6: Mean QoS values

where, $MinVal$ and $MaxVal$ are respectively the minimum and maximum recorded value in the system for that QoS metric. Note that, some of the QoS metrics have values which are interpreted inversely, i.e. the higher is the value, the lower is the quality. This includes execution time and price. Thus, the scaled value $NewQosVal$, for these type of QoS, is calculated as follows:

$$NewQosVal = 1 - (\frac{QosVal - MinVal}{MaxVal - MinVal}) \tag{7.9}$$

3. *Solving an equation system:* We assume that reputation and QoS are collinear. Thus, from the scaled data of Table 7.5, we define the equation system of m equations with 3 variables (QoS attributes) as follows:

$$R(S_k) = \begin{cases} T_1.X_1 + A_1.X_2 + P_1.X_3 &= R_1 \\ T_2.X_1 + A_2.X_2 + P_2.X_3 &= R_2 \\ T_3.X_1 + A_3.X_2 + P_3.X_3 &= R_3 \\ \vdots & \vdots \\ T_m.X_1 + A_m.X_2 + P_m.X_3 &= R_m \end{cases} \tag{7.10}$$

where, X_1, X_2 and X_3 are the variables, the coefficients T_i, A_i and P_i are the QoS values, and R_i is the reputation value ($R(S_i)$). Each equation in the system represents the relation between QoS values and the reputation of each service. Finally, we solve the equation system to find values of X_1, X_2 and X_3.

4. *Evaluating reputation:* Finally, Once the equation system is solved, we can find the reputation of the newcomer service $\hat{R}_i$ by direct application of the values of X_1, X_2 and X_3 in the following equation:

$$\hat{R}_i = T_i.X_1 + A_i.X_2 + P_i.X_3 \tag{7.11}$$

In case the equation system has no solution for the m equations, we eliminate from the system one equation, which includes the oldest assessed value of reputation, and we solve the equation-system again. We keep eliminating equations and solving the system till we find a solution, and m is still greater or equals 3. In case we could not get a solution for the equation system, we assess the reputation of the new web service as follows:

- *Construct the mean vector of similar-services' QoS*: For instance, column S_τ in Table 7.6 represents the mean vector for the QoS values in Table 7.5. And, column S_i represents the vector that holds QoS data of the new service.

- *Compute the reputation $\hat{R}_i$ of the new service S_i* by applying the following formula:

$$\hat{R}_i = \frac{R_\tau}{3} \times (\frac{T_i}{T_\tau} + \frac{A_i}{A_\tau} + \frac{P_i}{P_\tau}) \tag{7.12}$$

This formula stipulates that the ratio of the newcomer service reputation to the reputation of its similar services is equal to the average of the ratios of the QoS attributes of the newcomer service to those of its similar services.

7.4.3 Regression-based Reputation estimation

The third phase in the reputation bootstrapping technique is the construction of a multiple regression model, using QoS and reputation data of all long-standing web services. This model serves as an estimation of reputation for new services that has no previous interactions, nor they have known providers.

By definition, multiple regressions are statistical techniques used for predicting unknown Y values (dependent variable) corresponding to a set of X values (independent variables). In our study, the multiple regression is expected to give a model that could relate the reputation value of long-standing services to their QoS metrics, that is, we consider the dependent variable Y to represent the reputation of services as a function of multiple QoS attributes (independent variables) such as *response time, availability, throughput, latency, price,* etc. So, if we have n long-standing services in the system (S_j , $j = 1, 2, ..., n$), and each service S_j has a QoS vector $Q_{S_j} =< Q_{j,1}, Q_{j,2}, ..., Q_{j,k} >$ that holds k QoS metrics, and each service S_j has a reputation value $R(S_j)$ denoted R_j. The relationships between reputation (dependent variable) and QoS metrics (independent variables) can be expressed by the following equation:

$$\underbrace{\begin{pmatrix} Q_{1,1} & Q_{1,2} & \cdots & Q_{1,k} \\ Q_{2,1} & Q_{2,2} & \cdots & Q_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{n,1} & Q_{n,2} & \cdots & Q_{n,k} \end{pmatrix}}_{X} \underbrace{\begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \end{pmatrix}}_{\beta} + \underbrace{\begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}}_{\varepsilon} = \underbrace{\begin{pmatrix} R_1 \\ R_2 \\ \vdots \\ R_n \end{pmatrix}}_{Y} \tag{7.13}$$

Where :

- X is the design matrix that packs all regressors (predictors) $Q_{i,j}, i = 1, ..., n$ and $j = 1, ..., k$.
- β is the regression coefficient vector (called also slop vector).
- ε is the error vector. Error terms $\varepsilon_i, i = 1, .., n$ capture all the factors which influence the dependent variable $(R_i, i = 1, ..., n)$ other than regressors $(X_{i,j}, i = 1, ..., n$ and $j = 1, ..., k)$.

The multiple regression of the model can be simplified to:

$$R_i = \beta_1 Q_{i,1} + \beta_2 Q_{i,2} + ... + \beta_k Q_{i,k} + \varepsilon_i, \qquad i = 1, ..., n \tag{7.14}$$

Where,

- y_i is the response (estimated reputation) of the linear combination of the model terms.
- β_i $(i = 1, ..., k)$ represents the unknown coefficients.
- ε is the error term.

At the end of phase three, the system uses solved values of the unknown coefficients (β_i $(i = 1, ..., k)$, and the error term (ε), to estimate the reputation of the new comer services based on its initial QoS vector.

7.4.4 Evaluation of the bootstrapping Model

To evaluate the proposed bootstrapping technique, we conducted our experiment on a set of real web services data extracted from WSDream [200] and QWS [2] datasets. The aim of the experiment is to study the feasibility of the proposed technique, and to guarantee its efficiency in estimating reputation values of new-comer Web services. We focused on phase three (the construction of the multiple regression model), which is the worst (and somehow the most general) case. We evaluated the regression model regarding to its coefficient of determination (R^2)

Number	Quality	Description	Unit
1	Response time	Time taken to send a request and receive a response	ms
2	Availability	Number of successful invocations / total invocations	%
3	Throughput	Total Number of invocations for a given period of time	invocations/second
4	Successability	Number of response messages / number of request messages	%
5	Reliability	Number of error messages / total messages	%
6	Compliance	The extent to which a WSDL document follows WSDL specification	%
7	Best Practices	The extent to which a Web service follows WS-I Basic Profile	%
8	Latency	Time taken for the server to process a given request	ms
9	Documentation	Measure of documentation (i.e. description tags) in WSDL	%

Table 7.7: QoS metrics selected from QWS dataset

and its F-significance. Moreover, we tested the performance of the model by comparing estimated reputation values and reputation values assessed during a simulation, where we pretend that newcomer services are old in the system, and they have received feedback ratings from different users (both honest and malicious users are considered). Comparison results are reported by the Mean Absolute Error (MAE), and the Percentage-Error metrics.

Data description and preparation

WSDream dataset holds 5825 web service QoS data evaluated by 339 users in different geographical locations (we have chosen the dataset 2 in WSDream). The dataset holds 339*5825*2 (2: response time and throughput). QWS dataset holds 365 web services with 9 QoS metrics listed in Table 7.7.

To use a maximum number of QoS metrics with different monitored values of response time and throughput, we selected the services that belong to the two datasets. We matched web services based on their URIs, Names, and WSDL file size. We obtained 409 services that constitute the intersection set. Each service in this set has 7 fixed QoS metrics from QWS, and 2 QoS metrics (response time and throughput) that vary based on the observation of 339 users from WSDream. This final web service set is used for experimentation.

Feedback rating simulation

Due to the current limited availability of feedback rating data, many web service reputation management approaches (e.g., [101,102,120]) have used simulation for generating user feedback ratings for assessing service reputation values. Likewise,

Regression Statistics	
Multiple R	0.886815467
R Square	0.786441673
Adjusted R Square	0.781072888
Standard Error	0.272880252
Observations	368

ANOVA	df	SS	MS	F	Significance F
Regression	9	98.16965139	10.90773904	146.4841128	2.8767E-114
Residual	358	26.6579802	0.074463632		
Total	367	124.8276316			

	Coefficients	Standard Error	t Stat	P-value	Lower 95%	Upper 95%	Lower 95.0%	Upper 95.0%
Intercept (ε)	2.374640371	0.219162569	10.83506359	7.52464E-24	1.943632519	2.805648222	1.943632519	2.805648222
Response Time	-0.000346194	0.00529994	-0.065320349	0.09479554	-0.010769122	0.010076734	-0.010769122	0.010076734
Throughput	-0.000396292	0.000137447	-2.883247406	0.004173789	-0.000666596	-0.000125988	-0.000666596	-0.000125988
Availability	0.009825812	0.005457871	1.800301188	0.072654752	-0.000907706	0.02055933	-0.000907706	0.02055933
Successability	0.013471355	0.005074587	2.654670304	0.008292994	0.003491609	0.0234511	0.003491609	0.0234511
Reliability	-0.000504619	0.002832241	-0.178169433	0.85869069	-0.00607454	0.005065303	-0.00607454	0.005065303
Compliance	0.020737736	0.001661812	12.47899235	6.38821E-30	0.017469597	0.024005876	0.017469597	0.024005876
Best Practice	-0.005847026	0.002303954	-2.537822196	0.011577834	-0.010378011	-0.001316041	-0.010378011	-0.001316041
Latency	-0.000348913	9.14902E-05	-3.813661459	0.000161212	-0.000528838	-0.000168987	-0.000528838	-0.000168987
Documentation	0.007401319	0.000420714	17.59226209	2.2905E-50	0.006573936	0.008228701	0.006573936	0.008228701

Estimated reputation = 2.374640371 - (0.000346194 * Response Time) - (0.00396292 * Throughput) + (0.009825812 * Availability) + (0.013471355 * successability) - (0.000504619 * Reliability) + (0.020737736 * Compliance) - (0.005847026 * Best Practice) - (0.000348913 * Latency) + (0.007401319 * Documentation)

Figure 7.3: Output from the regression data analysis tool

we have built a Java program that simulates the interaction between the selected (409) web services and (339) users.

Each service has an actual performance (overall quality) level (from 1 to 10), denoted *PerfVal*, that represents in a scale of 10 how good is the overall quality provided by the service. PerfVal is calculated based on a utility function (i.e., a single scalar metric to quantify quality perception) of the delivered service, as suggested in [94]. However, in our work, we propose to calculate the utility function with the root mean square, which is a measure of the magnitude, of the scaled QoS metrics. Thus, *PerfVal* of service S_i is assessed as follows:

$$PerfVal(S_i) = 10 \times \sqrt{\frac{\sum_{j=1}^{k} Scal(Q_{i,j})^2}{k}} \tag{7.15}$$

Where, k is the number of used QoS metrics (Q_j, $j = 1, .., k$). And, $Scal(Q_{i,j})$ is the scaling function, which is defined by Eq. 7.16, if the quality is positive (i.e., the higher is the value the higher is the quality), and by 1 - the same formula otherwise.

$$Scal(Q_{i,j}) = \frac{Q_{i,j} - Min(Q_j)}{Max(Q_i) - Min(Q_j)} \tag{7.16}$$

where, $Min(Q_j)$ and $Max(Q_j)$ are respectively the minimum and maximum recorded values of the quality Q_j.

The program simulates two kinds of users: honest and malicious users. Honest users randomly rate a service based on its *PerfVal* within the interval

$[Max(0, PerfVal-2), Min(PerfVal+2, 10)]$, e.g, if *PerfVal*=7, honest feedback ratings could be 5, 6, 7, 8, and 9. the deviation with ±2 from *PerfVal* represents natural variation between user opinions. Malicious users randomly rate the same service outside the previous interval, always on scale of 10. For instance, if *PerfVal*=7, malicious feedback ratings could be 0, 1 ,2, 3, 4, and 10.

In this simulation, we consider 10% of users are malicious users. Finally, the reputation of web services are calculated as the mean of collected feedback ratings. The final reputation values are the average of 10 round-simulation results.

Multiple regression model building

We build the regression model using the QoS and reputation data of 90% of web services (Training set). The remaining 10% of services are used as service-test set (i.e., services considered as newcomers). We have used Microsoft Excel Regression data analysis tool to build our multiple regression model, because it provides a detailed analysis results about the regression model as it is shown in Figure 7.3. The same model is also generated by our Java program to estimate the reputation of the elements of the test set.

As we can see from Figure 7.3, the correlation coefficient (Multiple R=0.8869) indicates a positive relationship between reputation value and QoS data (where, 1 indicates a perfect positive relationship). We see also that $R^2 = 0.7864$ which means that 78.64% of the values fit the model.

Finally, since the significance-F (p-value) = 2.8767e-114 < 0.05, we conclude that the regression model is a significantly good fit (i.e, the calculated linear equation fits our data). For estimating web service reputation, we use the formula depicted in the bottom of Figure 7.3.

Reputation estimation and evaluation

We test the efficiency of the multiple regression model in the estimation of web service reputation, using test web services data (the remaining 10% of web services). We compare the estimation reputation value of each service obtained from the model, with the reputation value calculated from user feedback ratings collected during multi-round simulation. The **Mean Absolute Error** (MAE) between the estimated and calculated reputation values of the 41 test web services is equal to **0.2651**, with a **Percentage Error** (PE) of **4.656%**. A comparison between estimated and calculated reputation values of a sample of test services is shown in Figure 7.4. We have run multiple experiment rounds by varying the density of malicious users (i.e., rate of malicious users in the system). We have built regression models from the obtained results using the same training

Malicious Density	R^2	MAE	Percentage Error
10%	0.776797	0.29042	5.06319
20%	0.760624	0.29270	5.04946
30%	0.75072	0.26922	4.57920

Table 7.8: R^2, MEA and PE comparison by varying malicious density

set (90% of services). Moreover, we tested regression models using the same test service set. Table 7.8 lists the recorded R^2, MAE and Percentage Errors. As we can see, with the variation of malicious density, the regression models provide good results (e.g., R^2= 0.7507 with Malicious density = 30%). In addition, the obtained MAE and PE values indicate that there is a slight variation of the estimated reputation values. This variation is caused by the generated models.

Finally, we may safely use the estimated reputation value from the built regression model for bootstrapping the reputation of the newcomer web services. Because, even with greater deviation, the use of these bootstrapped reputation values is still better than assigning initial high (maximum), low (minimal), or any fixed (*e.g.,* average) reputation values.

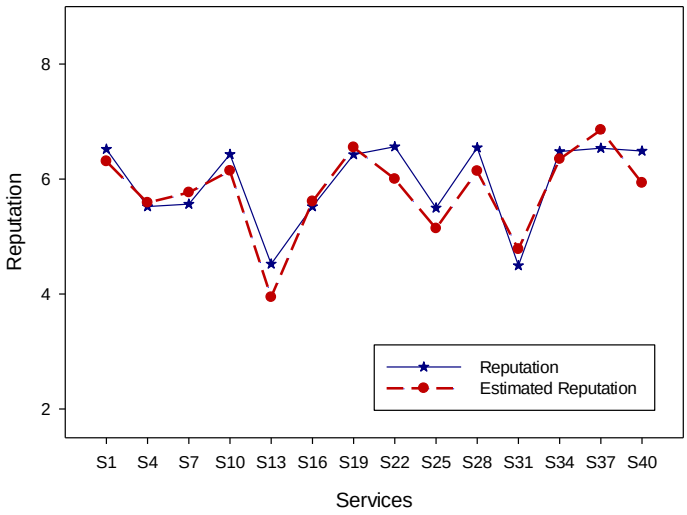

Figure 7.4: Calculated and estimated reputation of a sample of test services

7.5 Experiments

In this section, a series of experiments are conducted to evaluate the reliability of the proposed Reputation Management Model, and to show its effectiveness and performance. Firstly, we study the impact of users' subjective and malicious feedback ratings on the accuracy of the proposed model. Secondly, we investigate the impact of the included parameters on the performance of the assessment model. Finally, we conduct a comparison between the proposed model and different existing approaches.

7.5.1 Description

Due to the current limited availability of feedback rating data, many reputation management approaches such as [99, 101, 102, 176, 186] use simulation for model verification and performance evaluation. Likewise, we evaluate the performance of the proposed assessment model through simulation. We have built a concurrent Java application that simulates the interactions between Users, Web services and the Reputation Manager. The application is designed in a way that the behavior of web services is monitored, yet accurately captured. Hence, the application can simulate the behavior and feedback ratings of honest and malicious users accordingly. It assesses the reputation of web services according to the collected feedback ratings, aging factor and user's honesty (credibility). The variance between the assessed reputation scores of web services and their ideal (expected) reputation (represented numerically in the interval of $[0, 1]$) is a determinant factor for the validation of the proposed Reputation Management Model.

Table 7.9: Classes of simulated web services

WS Class	Reputation	Execution time (ms)	Description
C_1	[0.8 - 1]	[20-60]	Continuous high performance
C_2	[0-0.2]	[100-150]	Continuous low performance
C_3	[0.8 - 1] $\searrow$ [0 - 0.2]	[30-60] $\searrow$ [100-150]	High Performance then a degradation
C_4	[0 - 0.2] $\nearrow$ [0.8 - 1]	[100-150] $\nearrow$ [30-60]	Low performance then an enhancement
C_5	[0-1]	[20-150]	Oscillate performance

Table 7.10: Honest User Rates

WS class	Rating		Description
C1	[6-10]		
C2	[0-4]		Users rate randomly
C3	[6-10]	[0-4]	in the interval of ± 1
C4	[0-4]	[6-10]	of the RefVal
C5	[0-10] (±2)		

The programmed application holds a number of web service (*#Services*), and a number of users (*#Users*). We have simulated the interaction between these elements and the reputation manager during time slots. Each time slot is simulated as one day (*#Day*). A number of feedback rating transactions (*#TransactionsPerDay*) are issued from interactions between users and web services in each day. At the end of each time slot, the reputation manager calculates and updates the credibility factor of each user. Then, it assesses the reputation score of web services using the model presented in Section 7.3. For accurate results, the application uses a multi-round run (#Rounds) of simulation. The program lists the average of the assessed reputation scores of services in each class along with their ideal reputation values. The simulation classes of web services are described in Subsection 7.5.1. Then, user classes are presented in Subsection 7.5.1. After that, parameter tuning and the basic simulation algorithm are explained in Subsection 7.5.1. At last, performance evaluation metrics are detailed in Subsection 7.5.1 .

Web service classes

The quality of web service varies over time due to its existence in a dynamic environment where changes occur constantly [73]. We can distinguish five different classes of service behaviors [101]: a first class of services that maintain a high level of performance, a second class of services that maintain a low level of performance, a third class of "cheating" services that start with good performances then after a period of time, they degrade their performances. A forth class of services is composed of services which start with low performances and then they upgrade their performances. A fifth class includes services that have performances that oscillate.

We have implemented the five classes using Java's Randomization. In each

Table 7.11: Malicious User Rates

WS class	Rating	Description
C1	[0-6]	Users rate randomly by value that are »or «than the RefValue
C2	[5-10]	
C3	[0-6] [5-10]	
C4	[5-10] [0-6]	
C5	[0-10] ($\pm$3)	

class, the performance attribute value (*PerfVal*) represents the ideal reputation that correspond to the actual behavior of the web service. *PerfVal* ranges in the interval [0-1], where 0 denotes the lowest QoS level, and 1 represents a highest Qos level. The Response Time attribute ($ResT$) indicates web service's maximum response time. Table 7.9 lists the expected parameters (Ideal reputation (*PerfVal*), and Execution time ($ResT$)) of each class, where:

- Class C_1: The first class simulates the behavior of services that exhibit and maintain a continuous high performance (High Qos). The ideal reputation of such service ranges between 0.8 and 1. The response time of an instance ranges between 20 and 60 milliseconds.

- Class C_2: The second class simulates the behavior of services that exhibit and maintain a continuous low performance (Low Qos). The ideal reputation of an instance varies between 0 and 0.2. The response time of an instance ranges between 100 to 150 milliseconds.

- Class C_3: The third class simulates the behavior of services that exhibit a continuous high performance during the first half of the simulation time, with a response time that ranges between 20 and 60 milliseconds. Then, these services exhibit low performances (i.e. a performance degradation occurs) during the second half of the simulation time, with a response time that ranges between 100 and 150 milliseconds. The ideal reputation values of these services range accordingly in two intervals; between 0.8 and 1 in the first half of simulation time, and between 0 and 0.2 in the second half.

- Class C_4: This class simulates the behavior of services that exhibit a low performance during the first half of the simulation time with a response time that ranges between 100 and 150 milliseconds. Then, these services exhibit a high performance in the second half of the simulation time with a

Table 7.12: Simulation parameters

Parameter	Values
#Services	500 (100 per class)
#Users	1000
#Days	100
#TransactionsPerDay	10000 (2000 per class)
#Rounds	10
#HUserDensity	?% (variable)

response time that ranges between 20 and 60 milliseconds. The ideal reputation values of these services range accordingly in two intervals; between 0 and 0.2 in the first half of simulation time, and between 0.8 and 1 in the remaining period.

- Class C_5:This class simulates web services with an oscillating performance. The response time ranges between 20 and 150 milliseconds. The ideal reputation of an instance varies between 0 and 1.

These classes group all possible service behaviors from providers [101], which ensures that experiment samples are representative to real world environment.

Users Classes

To study the effect of users' credibility on the reputation assessment model, we developed two classes that imitate honest and malicious (dishonest) users; each class simulates the behavior of a user during her/his interaction with the web services and the reputation manager. Instances of honest users send a feedback that is approximate by ± 2 from the expected reputation value (i.e, a honest user sends a rating feedback which is equal to the ideal value or close to it). Instances of malicious user class produce feedback rating that differ at least by ± 3 from the expected value (i.e subjective feedback rates are always far from the expected (ideal) value). Tables 7.10 and 7.11 respectively summarize the feedback ratings generated by honest and malicious users for each class of service.

Tuning

The pseudo code in Algorithm 6 represents the basic flow of one round of the conducted simulation. We simulate a system with 500 web services. We create

Algorithm 6 Pseudo code of the conducted simulation

Input: *#HUserDencity*

Begin

```
1: #Days = 100 ;
2: #services = 500;
3: #TransactionsPerDay = 10000 ;
4: Create services(#Services);
5: Create users(#HUserDenity);
6: for simDay = 1 TO #Days do
7:     Generate_Ideal_Reputation_for_Services();
8:     for (class = 1 TO 5) do
9:         for (i=0 TO (#TransactionsPerDay/5)) do
10:            user = Select_random_user();
11:            service = Select_random_service_in_class(class);
12:            simulate_user_service_interactions(user, service);
13:            add_new_feedback_rating(user, service, simDay);
14:        end for
15:        evaluate_users_credibility();
16:        evaluate_daily_ideal_service_reputation();
17:        assess_daily_reputation_from_feedback_ratings();
18:        save_data();
19:    end for
20: end for
21: evaluate_results();
```

End

100 instances from each class of service. Then we create 1000 honest and malicious users according to the honest user density (#HUserDensity). For each time slot, which varies from 1 to the total number of simulated time slots (#Days), we generate the ideal reputation of each service instance according to its service class. Then, we simulate the transactions (interactions) between randomly selected users and randomly selected services. Transactions are conducted simultaneously. After each transaction, the program stores the feedback rating generated by the involved user. At the end of each time slot, the program updates user credibility scores, and it assesses daily the reputation score of each web service. The program at the end of all the transaction, measures the performance and prints results. Table 7.12 summarizes the parameters applied during the simulation.

Performance metrics

Performance metrics can be classified into statistical accuracy and decision-support accuracy metrics [26, 71]. We adopt one metric from each class to evaluate the performance of the proposed reputation assessment model. *Mean Absolute Error* (MAE) is a representative metric of a statistical accuracy measure. The MAE metric is defined as follows:

$$MAE = \frac{\sum_{k=1}^{n} |Rep_a(WS_k) - Rep_i(WS_k)|}{N} \tag{7.17}$$

where, $Rep_a(WS_k)$ and $Rep_i(WS_k)$ are the assessed reputation score and the ideal reputation score of service WS_k respectively, and N is the number of web services.

F-Measure is the second used metric. It is a representative metric of the decision-support accuracy measure. *FMeasure* (or F-score) combines *precision* and *recall* metrics into a single value that determines how effectively the Reputation Management Model assesses precisely the reputation of web services, and handles correctly subjectivity and maliciousness of users' activity. First, Precision is defined in this context via *Normalized Mean Absolute Error* (NMAE) as follows :

$$precision = 1 - NMAE \tag{7.18}$$

where

$$NMAE = \frac{MAE}{r_{max} - r_{min}} \tag{7.19}$$

where r_{max} and r_{min} are the maximum and the minimum rates respectively. *Precision* values range between 0 and 1, and decrease with the increase in NMAE. *Recall* is defined as the ratio of correctly evaluated services denoted Nbr_{cs} to total number of services (N). *Recall* represents the probability that a service reputation is correctly evaluated. It is defined as follows:

$$Recall = \frac{Nbr_{cs}}{N} \tag{7.20}$$

F-Measure is defined based on precision and recall as follows:

$$F\text{-}Measure = \frac{2 \times recall \times precision}{recall + precision} \tag{7.21}$$

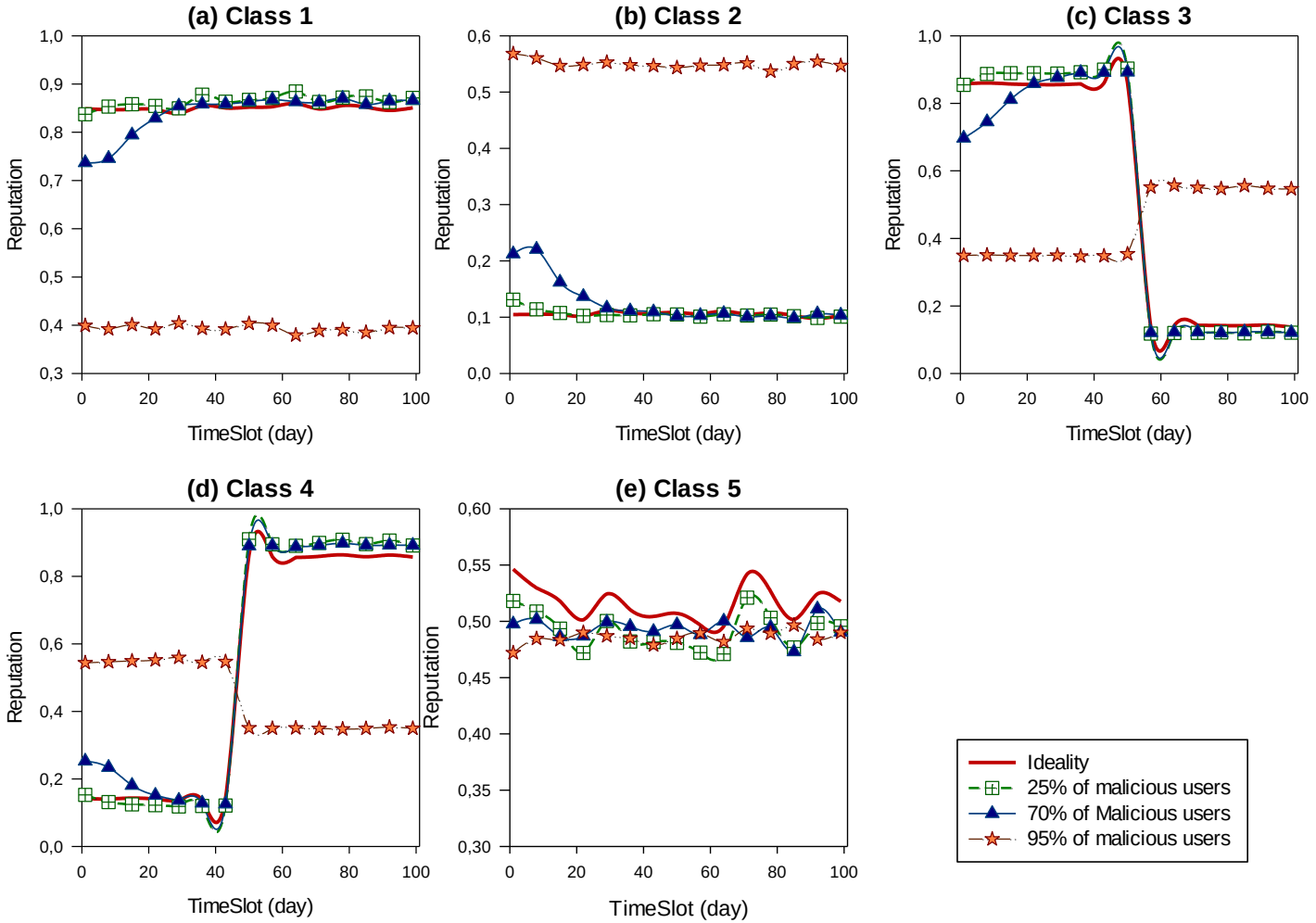

Figure 7.5: Ideal and assessed reputations with 25%, 70% and 95% of malicious user density

Java-based simulator application

Figure 7.6 depicts the graphical user interface of the implemented simulator. The user can set different simulation parameters, and then run the simulation. Results of each class are depicted in a separate graph. Considered metrics are plotted in separate graphs. Moreover, generated data are stored in specific files.

Figure 7.6: Graphical used interface of the Java-based simulator

7.5.2 Reputation with varying maliciousness density

The ultimate goal of this approach is to accurately assess the reputation of web services even though unfair feedback ratings collected from malicious users are

considerable. In this section, we show results of first instances of our experiment. We fixed simulation parameters to the values listed in Table 7.12, with a variation of maliciousness density. This density represents the rate (i.e., percentage) of malicious users in the system.

Figure 7.5 shows the ideal versus the assessed reputation scores obtained with three instances of maliciousness density (labeled 25%, 70% and 95%). In that figure, plots (a) through (e) are associated to the five web service classes described previously (Subsection 7.5.1). For each class, we obtain the plotted reputation values as the geometric mean of its service reputation scores (100 service per class). Yet, each service reputation score is the geometric mean of 10 simulation-round values.

First, we can see that with 25% of maliciousness density, the assessed reputation values in the five classes are almost equal to the ideal reputation values (showed by the solid line). These results are explained by the fact that honest users outnumber dishonest users, which permitted a successful user credibility evaluation and reputation assessment. The slightly deviation occurred between the assessed and ideal reputation is natural, because it reflects the differences in opinions between honest users.

Second, we observe that, with malicious user density that equates to 70%, the assessed reputation is below the ideal reputation in the first third of evaluation period (for time slot 1 to 30). Then, for the rest of the evaluation period, the assessed reputation becomes fairly and close to the ideal reputation (i.e., the assessed reputations converge to the ideal values). We note here that dishonest users outnumber honest users, which influences negatively on the number of fair feedback ratings, the evaluation of user credibility, and hence on the assessed reputation at the beginning of the evaluation. However, by the accumulation of fair and unfair feedback ratings, the model becomes able to distinguish between malicious and honest users based on their credibilities. Therefore, the system neutralizes the effect of unfair feedback ratings on the assessed reputation by penalizing suspicious users with a credibility lower than 0.5.

Third, experiments with malicious density equal to 95% shows that the assessed reputations are significantly deviating from the original reputation values. The model is unable to assess correctly reputation due the very high number of malicious users. However, Whitby *et al.* [178] and Malik *et al.* [101] claim that such high number of malicious user density in real world is unrealistic and much lower rate in real world applications should be expected.

Subsequently, we may conclude that the proposed assessment model is able

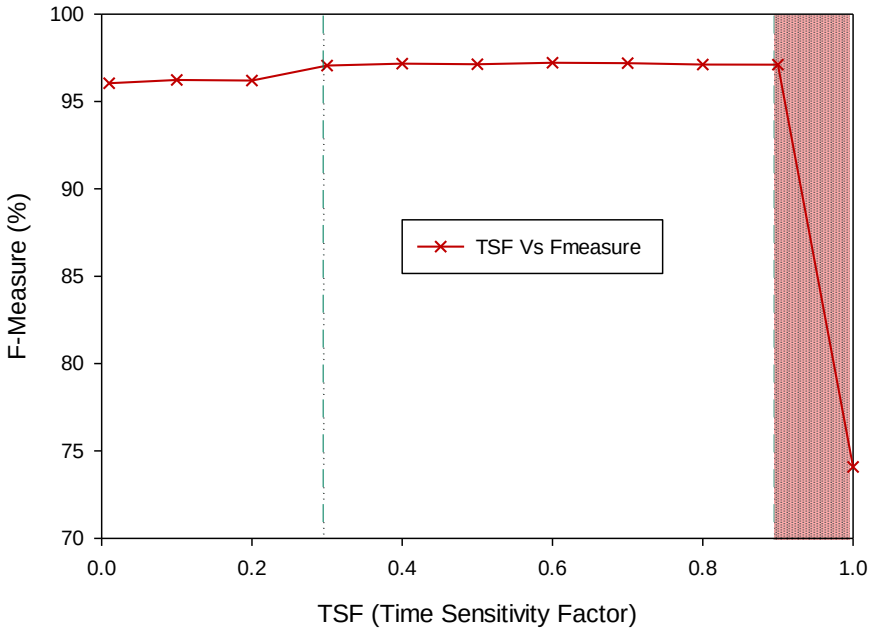

Figure 7.7: Effect of Time Sensitivity Factor on the F-Measure

to accurately assess the reputation of web services even with the presence of high malicious rates (up to 70%) in the reputation management system.

7.5.3 Impact of time sensitivity factor

In the second instance of simulation runs, we have studied the impact of the Time Sensitivity Factor (TSF or λ) on the performance of the proposed assessment model. We have varied the value of λ from 0.1 to 1 with a step value of 0.1. We fixed the number of time-slots to 1000 ($\#days = 1000$).

Figure 7.7 depicts the effect of λ on the global F-Measure. Values of global F-Measure are geometric means of the F-measure values of the five classes which are assessed using Eq.7.21. Note that all used scores are the mean of 10 simulation round values.

The figure shows that: (i) F-measure is slightly increased when λ varies from 0.1 to 0.3. (ii) F-measure is steady with the top value when λ varies from 0.3 to 0.9. (iii) However, F-measure considerably decreases when λ varies from 0.9 to 1. In conclusion, The best performances of our reputation assessment model is when λ ranges in the interval [0.3, 0.9].

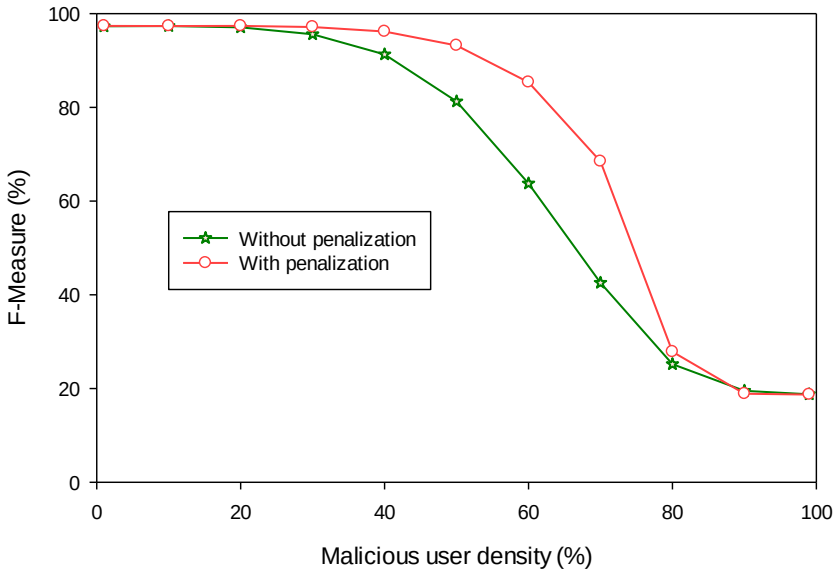

Figure 7.8: Effect of penalization mechanism on F-Measure performances

7.5.4 Effect of the penalization mechanism

In the third instance of simulation runs, we investigated how the penalization mechanism affects the assessment of reputation scores. We fixed the simulation parameter with the values listed in Table 7.12, and varying maliciousness density from 0% to 100%. In each run, we assess service reputation scores in two manners: (i) applying the penalization mechanism, and (ii) without application of the penalization mechanism. Figure 7.8 shows the obtained results for F-measure. From the figure, we observe that: First, when malicious user density varies from 0% to 20%, The performance of the model with the application of the penalization mechanism are approximately equal to the performances of the model without application of the penalization mechanism. This is interpreted by the low impact of unfair feedback ratings (low number of malicious users) on the assessed reputation scores (i.e., the value of unfair feedback ratings multiplied by their user credibilities is neglected in comparison to the value of fair feedback ratings multiplied by their user credibilities).

Second, when malicious user density varies from 20% to 90%, the performances of the model with penalization application are enhanced (up to 20% when malicious density are in the interval [50%-70%]) relatively to the performances of the model without penalization application. Thus, the model neutralizes effectively the impact of unfair feedback ratings from the assessed reputation scores.

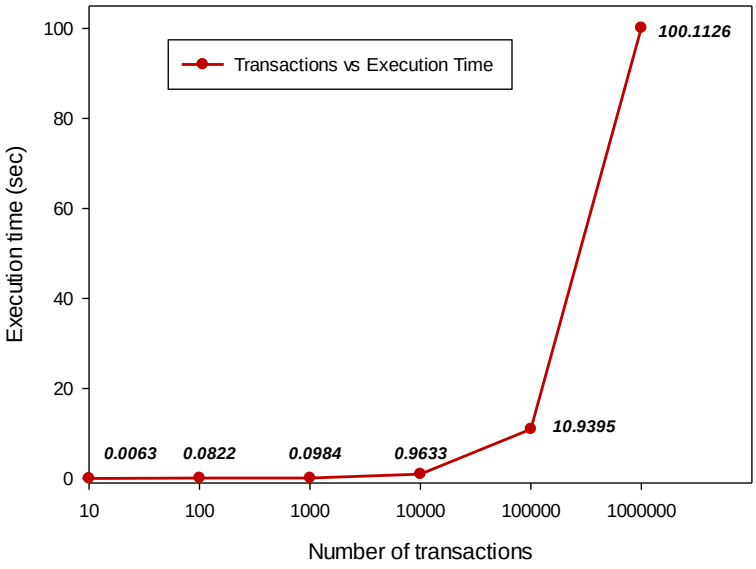

Figure 7.9: Execution time versus variation of transaction numbers

Third, the model with and without penalization application is unable to effectively assess the web service reputation scores when the number of malicious users is important, thus the number of unfair feedback ratings. Fortunately, according to Whitby *et al.* [178], these malicious densities is unrealistic in real world settings and much lower densities are expected.

Finally, we draw the conclusion that the application of the penalization mechanism significantly increases the performances of the proposed reputation assessment model.

7.5.5 Execution time performance

We present in this section our measurement of the execution time for processing user-service transactions, with the assessment and updating of service reputation values. User-service transactions represent the interaction between a user and a web service. It represents the process of selecting, consuming and sending a feedback rating the web service.

Figure 7.9 shows the execution time taken by the system to process different numbers of transactions. For these measurements, we started different runs on a single machine, Intel(R) Core(TM) i7-3537 CPU @ 2.00GHz With 8 GB RAM running under windows 8. We removed unnecessary processes from the operating system as much as possible to ensure reliable measurement. The experiments are repeated 10 times under the same settings. The values shown in the figure

represent the geometric means of the 10 trials.

For instance, the system can assess 100 times the reputation scores of 500 services from 10000 feedback transactions per time slot in 11 seconds. This value is good indicator about the performance of the model in term of processing time.

7.5.6 Performance comparison

In this section, we compare the performance of our proposed model with the following centralized reputation assessment approaches:

1. The normal approach (labeled Normal): This is the conventional approach where the reputation is assessed as the mean of collected feedback ratings, without considering any coefficient such as the time sensitivity factor and the user credibility factor.

2. The approach proposed by Wang *et al.* [176] (labeled Wang et al): The approach assesses the reputation score $q(s_j)$ of service S_j as follows:

$$q(s_j) = \frac{1}{n}\sum_{i=1}^{n} r_i$$

where r_i represents the i-th feedback rating, n (n=1,2, ...) is the number of feedback ratings. Note the approach assesses reputation values using only pure feedback ratings (fair ratings or adjusted malicious ratings), because the approach applies a malicious feedback ratings prevention scheme based on the Cumulative Sum Method (CUSUM). The CUSUM monitors n feedback ratings sample interval. For each sample interval, they assign a score $Z(y_i)$ which is assessed as follows :

$$Z(y_i) = \frac{\mu_1 - \mu_0}{\sigma^2}(y_i - \frac{\mu_1 - \mu_0}{2})$$

where, rating feedback sample intervals represented by $\{y_1, y_2, ...\}$ the variable $y_i (y_i = \sum_{i=1}^{m} r_i)$ $(i \leq j \leq n)$ $(m = 1, 2, ...)$, and μ_0 and μ_1 represent the mean feedback rating traffic before and after the change. When a sample interval is available, the CUSUM f_i is updated as follows :

$$f_i = \max(f_{i-1} + Z(y_i), 0)$$

if $f_i \geq h$ then a positive shift occurs in the n-th sample which means that there is an abnormal detection point (presence of malicious feedback rating). In our implementation of this scheme we set h to 0.7 based on the authors experiment settings.

3. The approach proposed by Mekouar *et al.* [107] (labeled TrustWS): the authors propose to assess the reputation of web services as the difference between positive and negative feedback ratings divided by the sum of both. Reputation is set to 0 when the sum of feedback ratings equates to 0. This approach do not include time sensitivity factor nor the credibility of users for reputation assessment.

Performance comparison with a fixed density of malicious users

We present in Figure 7.10 a comparison between the reputation scores assessed by our model and reputations scores obtained by the three approaches cited above, using 25 % as malicious user density parameter. The reference baseline of this comparison is the ideal reputation scores that are presented by a simple solid line in the different plots (a-e).

Form the figure, we can see that our reputation scores are closer to the ideal reputation than scores obtained by the other approaches for the five different service classes.

It is also worth mentioning that reputation scores assessed by the approach of Wang *et al.* are also steady and fairly close to the ideal reputation scores for the first, second and the fifth class of services. However, the "prevention scheme" considers fair ratings received after an abrupt change of service QoS as malicious feedback rating, since they produce positive shifts detected by CUSUM. Therefore, the approach shows a significant deviation from the ideal reputation in the third and fourth service classes, during the second half of evaluation period as depicted in graph (c) and (d). This limitation is highlighted by the authors in their paper [176] (Section 5.4).

Moreover, TrustWS successfully assesses the reputation of services that maintain steady high Qos as it is depicted in Figure 7.10-(a), in the first half of graph (c), and the second half of graph (d).

In addition, reputation values assessed by the normal method converge to the ideal reputation for services with steady low Qos as we can see in graph (b). Unfortunately, the non-inclusion of credibility and time sensitivity factors affects the performance of TrustWS and the normal method to assess correctly reputation scores in the other cases.

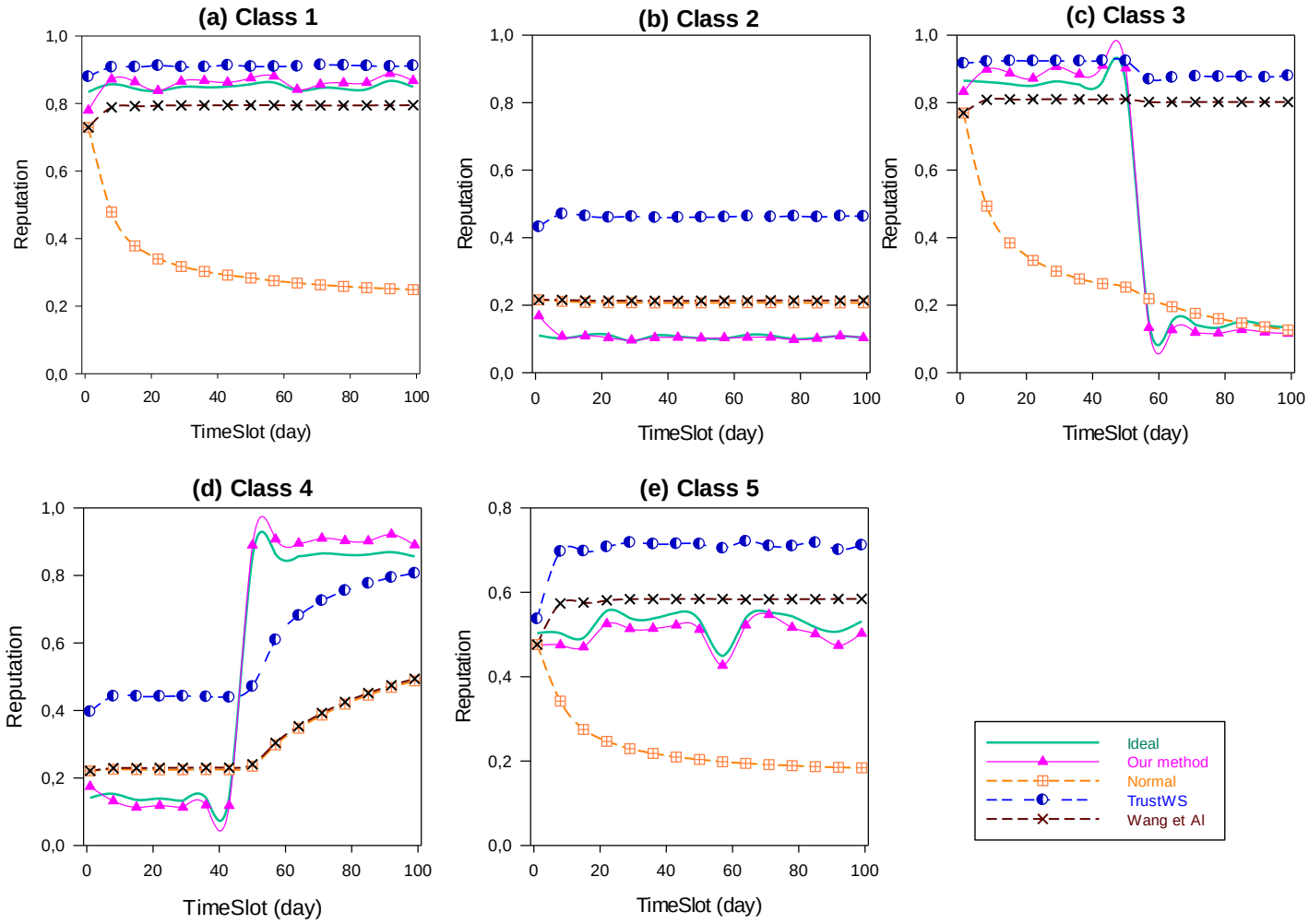

Figure 7.10: Ideal and compared assessed reputations with 25% of malicious user density

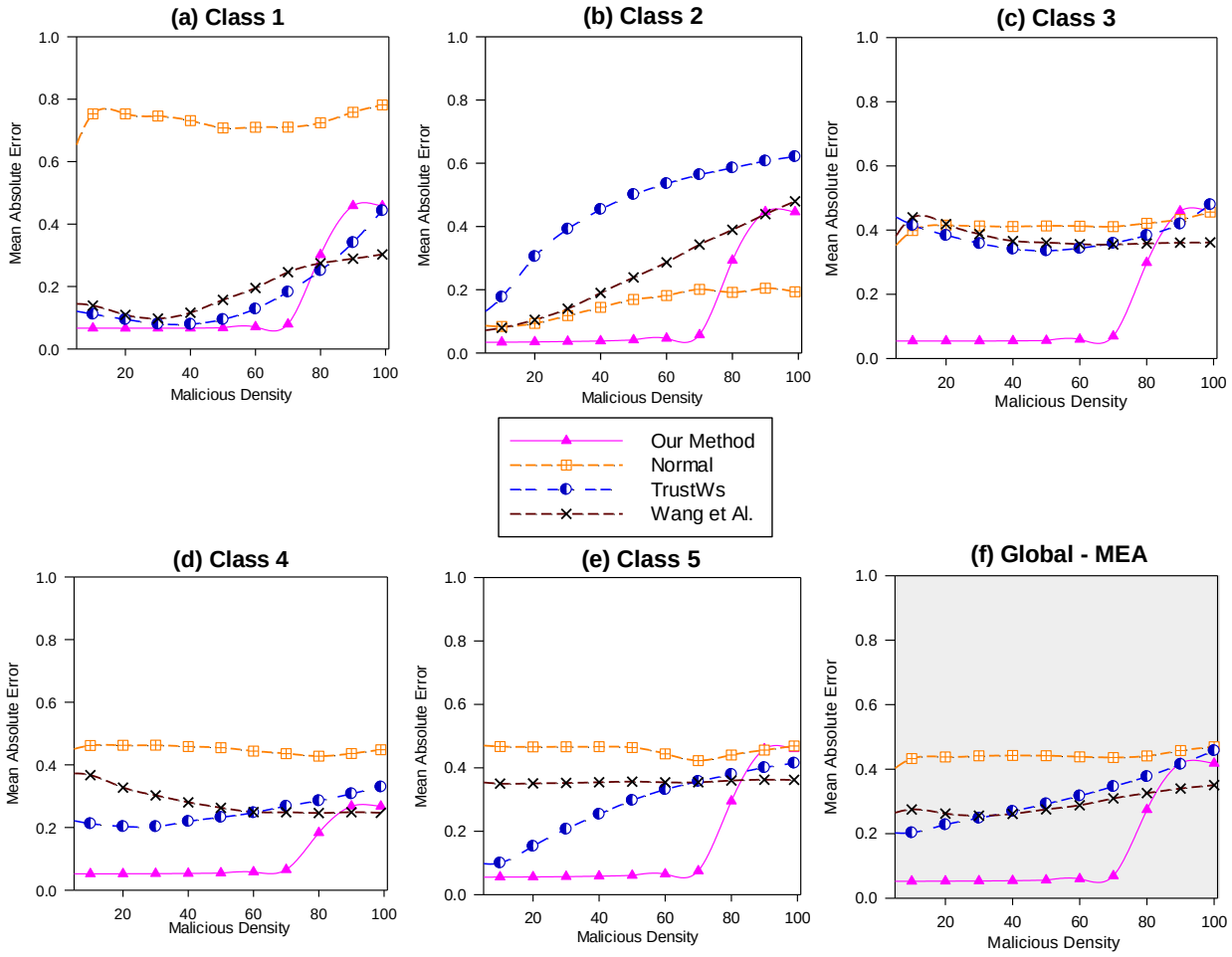

Figure 7.11: Mean Absolute Error comparison with the alteration of malicious users density (a smaller MAE means a better performance)

Table 7.13: Mean Absolute Error and F-Measure values comparison

Malicious density	Method	Mean Absolute Error						F-Measure					
		Class 1	*Class 2*	*Class 3*	*Class 4*	*Class 5*	Global	*Class 1*	*Class 2*	*Class 3*	*Class 4*	*Class 5*	Global
1 %	Our Method	0.067	0,034	0,055	0,055	0,052	0,052	96.543	98.294	97.177	97.153	97.077	97.249
	Normal	0.536	0,090	0,300	0,473	0,437	0,367	39.987	95.274	54.414	45.996	35.706	54.276
	TrustWs	0.128	0,105	0,465	0,105	0,230	0,207	89.785	94.467	46.202	93.348	64.553	77.671
	Wang et Al.	0.144	0,067	0,307	0,358	0,371	0,249	83.580	96.537	48.467	22.198	41.800	58.516
10 %	Our Method	0,067	0,034	0,054	0,055	0,052	0,052	96.560	98.269	97.188	97.140	97.011	97.234
	Normal	0,753	0,084	0,399	0,467	0,462	0,433	6.314	95.631	54.104	46.107	34.409	47.313
	TrustWs	0,111	0,177	0,415	0,100	0,212	0,203	93.369	79.173	47.042	90.939	69.324	75.969
	Wang et Al.	0,139	0,080	0,440	0,350	0,367	0,275	85.843	95.489	47.336	11.365	44.764	56.959
20%	Our Method	0,066	0,035	0,054	0,056	0,052	0,053	96.568	98.226	97.187	97.116	96.877	97.195
	Normal	0,753	0,094	0,415	0,466	0,462	0,438	6.382	95.083	50.878	46.150	34.474	46.593
	TrustWs	0,094	0,305	0,384	0,153	0,203	0,228	94.835	44.269	47.939	79.794	70.332	67.434
	Wang et Al.	0,109	0,105	0,419	0,351	0,327	0,262	91.768	93.130	48.113	11.606	47.534	58.430
30%	Our Method	0,066	0,036	0,054	0,057	0,053	0,053	96.561	98.153	97.191	97.067	96.720	97.138
	Normal	0,746	0,118	0,413	0,466	0,463	0,441	8.156	93.024	52.263	46.079	34.396	46.784
	TrustWs	0,080	0,392	0,358	0,207	0,203	0,248	95.725	15.022	48.603	66.496	68.898	58.949
	Wang et Al.	0,097	0,140	0,388	0,352	0,302	0,256	93.617	90.160	48.142	14.043	50.213	59.235
40%	Our method	0,067	0,038	0,055	0,058	0,053	0,054	96.549	98.058	97.162	96.986	96.540	97.059
	Normal	0,731	0,144	0,411	0,467	0,459	0,442	11.348	85.683	53.144	45.944	34.696	46.163
	TrustWs	0,079	0,454	0,341	0,254	0,219	0,269	95.719	7.592	48.689	56.830	63.965	54.559
	Wang et Al.	0,116	0,190	0,366	0,354	0,280	0,261	90.427	76.879	47.150	22.196	50.429	57.416
50%	Our method	0,068	0,041	0,056	0,060	0,054	0,056	96.461	97.870	97.084	96.858	96.184	96.892
	Normal	0,708	0,169	0,413	0,464	0,455	0,442	12.272	83.693	50.786	45.231	34.816	45.359
	TrustWs	0,094	0,501	0,335	0,297	0,233	0,292	94.044	3.483	48.095	48.754	60.002	50.876
	Wang et Al.	0,157	0,239	0,361	0,356	0,263	0,275	81.880	69.816	42.922	25.423	49.428	53.894
60%	Our method	0,070	0,046	0,059	0,065	0,058	0,060	95.659	87.075	96.418	96.197	93.122	93.694
	Normal	0,710	0,182	0,412	0,444	0,444	0,438	10.325	57.172	47.809	40.710	21.832	35.569
	TrustWs	0,128	0,536	0,343	0,331	0,247	0,317	87.440	1.515	45.344	43.182	52.415	45.979
	Wang et Al.	0,195	0,286	0,356	0,354	0,249	0,288	72.752	28.901	33.093	21.946	43.023	39.943
70%	Our method	0,079	0,057	0,069	0,075	0,065	0,069	94.440	86.751	95.076	95.129	91.351	92.550
	Normal	0,711	0,201	0,411	0,423	0,436	0,436	6.353	57.238	40.423	36.380	21.702	32.419
	TrustWs	0,182	0,564	0,359	0,357	0,268	0,346	76.037	0.703	38.591	40.505	48.411	40.849
	Wang et Al.	0,246	0,343	0,355	0,354	0,249	0,309	58.988	22.771	18.616	20.644	42.297	32.663
80%	Our method	0,302	0,293	0,298	0,295	0,183	0,274	41.380	48.066	41.706	43.369	66.896	48.283
	Normal	0,724	0,192	0,421	0,440	0,428	0,441	2.678	70.761	40.732	34.201	36.214	36.917
	TrustWs	0,251	0,585	0,383	0,379	0,286	0,377	58.903	0.370	23.020	36.491	48.170	33.391
	Wang et Al.	0,275	0,389	0,358	0,360	0,246	0,326	50.355	4.816	9.786	8.817	45.534	23.861
90%	Our method	0,459	0,446	0,459	0,459	0,268	0,418	23.864	4.316	3.138	3.222	50.600	17.028
	Normal	0,758	0,205	0,433	0,456	0,436	0,458	0.710	69.930	43.658	34.829	35.700	36.965
	TrustWs	0,341	0,607	0,419	0,401	0,308	0,415	39.217	0.113	6.145	33.303	44.996	24.755
	Wang et Al.	0,289	0,439	0,361	0,362	0,249	0,340	47.745	0.211	20.415	15.443	45.300	25.823

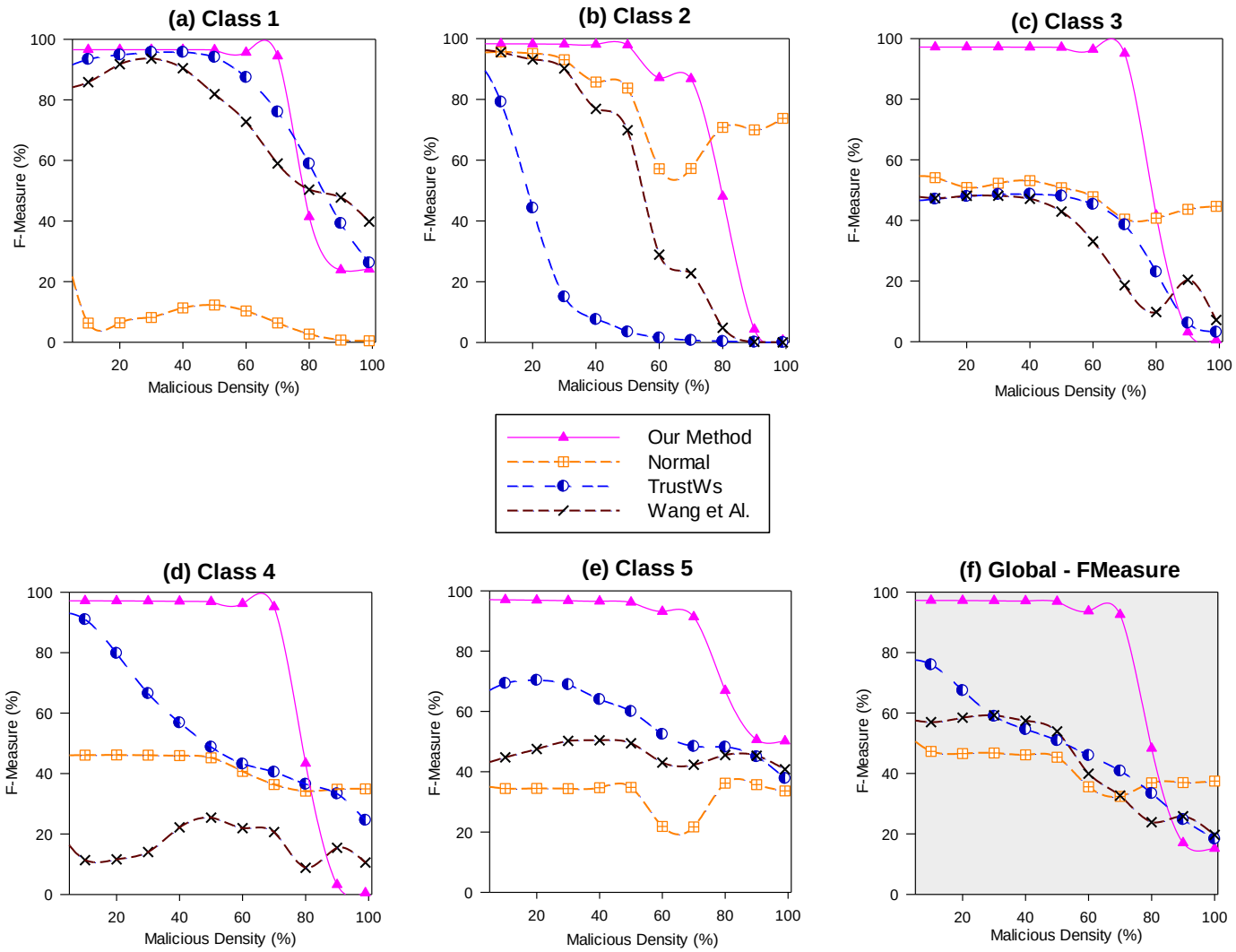

Figure 7.12: F-Measure performance comparison with the alteration of malicious users density

Performance comparison with alteration of malicious user density

For farther performances comparison between the four approaches, we conducted other simulation runs, using the parameters of Table 7.12 and varying the malicious user density. Table 7.13 shows the MAE and F-Measure results of different reputation approaches. Each experiment is run 10 times and the averages of MEA and F-Measure are reported. For better visibility, we plotted MEA and F-Measure results, respectively, in Figures 7.11 and 7.12 . Experimental results show the following:

- Under different settings, our reputation assessment model obtains the smaller MAE and the higher F-Measure values (up to 97%) consistently, within the interval [0%, 70%] of malicious user density. These results indicate a better accuracy of the reputation assessment.
- The approach of Wang *et al.* obtains smaller MAE and higher F-Measure values when services maintain stable QoS. However, reputation scores diverge from the ideal values when service QoS is quickly and significantly upgraded or degraded (when services suddenly change from good to bad or inversely). This divergence is presented by the increase of MAE and the decrease of F-Measure as with services of class 3 and 4. Hence, the global MEA and the global F-Measure are negatively influenced by this divergence, as it is depicted respectively in Figures 7.11-(f) and 7.12-(f).
- Even that it does not include credibility and time sensitivity factors, TrustWS measures accurately the reputation of services in the first class (services with consistent high QoS), as depicted by high F-Measure scores in Figure 7.12-(a). Nevertheless, for the same reason, it fails to assess accurate reputation scores for the remaining classes, as it is shown in plots (b-f) in Figure 7.12.
- Likewise to TrustWS, the Normal approach assesses correctly the reputation of the second class services (i.e., service with steady low Qos), as the associated MAE and F-Measure indicate in the figures. However, this naive method fails for the other classes.

7.5.7 Limitations

The experiment shows that the proposed reputation assessment model provides sound and accurate measurement of web service reputation. Nonetheless, there are few limitations that we list below:

- Our model is not adequate for assessing reputation of old services with a very low number of feedback ratings, or services with a long discontinuous

feedback rating. The more feedback ratings the system collects, the more accurate results the system provides.

- The use of IP addresses to identify users raises a problem when dynamic IP addresses are used. This means that feedbacks sent by the same user with different IP addresses are not recognized to belong to the same user, and hence, threats to evaluate user credibilities and reputation scores are raised.

However, to guarantee, with our reputation assessment model, a better accuracy and efficiency, it is recommended that first feedback ratings in the system have to be collected from trusted users.

7.6 Summary

In this chapter, we have presented a framework for the management of web service reputation. The framework describes how users feedbacks are collected, processed and aggregated. We have described the proposed reputation assessment model. The model include many factors such as user credibility, time sensitivity and majority rating for instance.

One of the novel contributions of this model, is the inclusion of the penalization factor, which discards unfair feedback rating from been included in the reputation assessment.

In addition, we have introduced a reputation bootstrapping model, which estimates reputation values of newcomer web services based on their initial QoS. This model is built upon three phases: (1) estimating reputation from service provider, (2) estimating reputation from similar services, and (3) estimating reputation using a regression-based model that evaluates reputation from the whole service pool. We have partially validated the proposed reputation bootstrapping mechanism.

Moreover, have conducted two extensive experiments to validate the proposed reputation bootstrapping and assessment models. We have reported experiment results, demonstrating the performance and effectiveness of these contributions.

Chapter 8

Conclusion and Future Work

Contents

This chapter summarizes the contribution made in this thesis, and outlines directions for future work.

8.1 Summary

This thesis addressed some challenges related to the orchestration of reliable web services. We mean by reliability the ability that the system (Web services and the whole orchestration) can complete its tasks whenever it is invoked. In the web service context, reliability for service designers can be considered as the ability to trust that services offered by other providers fulfills functional and non-functional requirements and work as expected.

Based on several researcher opinions (e.g., [73]), the reliability of the whole web services orchestration depends mainly on reliabilities of its atomic web services. Therefore, the selection of reliable web services is a key solution to ensure reliability of the whole service orchestration. However, faults and incidents are inevitable in failure-prone environments, and initial selected services may obstruct the execution of the orchestration if one of these services get defected. Thus, the selection of service substitutes and the design of recovery plans is also a crucial topic to ensure reliability in web services orchestration. In addition, a reliability analysis has to be conducted over different phases in the orchestration's life-cycle, which allow a permanent control of reliability and QoS satisfaction.

Moreover, when selecting services, either at design time or as substitutes at maintenance phase, trust and reputation may be considered as an overall reliability indicator for web services and their providers, because often a reputable service and a trustworthy provider are experienced to complete their tasks and achieves a certain level of reliability.

In this thesis, we addressed some issues related to these points by proposing the following :

Firstly, we presented in Chapter 4 the life cycle of a web service composition that supports reliability. We have mentioned that a continuous reliability analysis and verification is required during different life-cycle phases. Moreover, a dynamic web service list have to be provided by a smart registry (web service recommendation system) for performing adaptation and self reconfiguration. In the second section of Chapter 4, we proposed a generic architecture for conducting reliable web service orchestrations. The architecture assists in ensuring reliable web services orchestrations. We described the main components of this architecture, and we explained the interactions between these elements during different orchestration design and execution phases.

Secondly, in chapter 5, we proposed an approach for the measurement of web services similarity based on their WSDL interfaces. In fact, the study of similarity was a necessary step before proposing any service and substitute selection approach, because the study of similarity between web services reveals similarity, substitutability and composability relationships between services. In addition, it provides a practical solution that allows the reuse of large number of web services freely available on the Internet by facilitating the matchmaking process between service description and seekers' queries.

In general, the proposed approach is parameterized (customized) by different kinds of weighted scores and the use of multiple metrics that have been successfully applied in information retrieval's problems. Weighted scores are measured by analyzing WSDL descriptions of Web services interfaces. The proposed similarity measurement process starts by calculating similarity between service names, operations, input/output messages, parameters, and at last compares the documentation. It addresses at the same time the lexical and semantic similarities between identifiers. It makes schema matching for comparing message structures and complex XML schema types. The similarity scores issued from the measurement process can determine whether the compared web service are substitutes to each other, or only a subset of operations which are substitutes to each other, or even more, the two web services are completely different.

We implemented this approach by developing a prototype, that we coin WS-SIM, that assesses the similarity between two services, or a collection of services. The prototype could be used by designers as a standalone application, as it could be integrated in more capable softwares as it is provided also as a java API. The approach was experimented and validated using a collection or real web services.

Thirdly, we proposed in chapter 6 an approach for the identification of web service substitutes. The approach uses the WSDL (Web Service Description Language) interface description of the failed web service (partner link) to discover, from a service pool, a set of web service candidates that offer the same or related functionalities. The set is then refined using a filtering algorithm that exploits the similarity assessment technique proposed in Chapter 5 in order to keep only web services that have similarity relationships between them. Using the same similarity assessment technique, a similarity matrix between the input and the output of web service operations is built, and then it is exploited to create a concept lattice using Formal Concept Analysis. A process of refinements is applied on the similarity matrix and its associated formal concepts to build the final concepts and lattices. The navigation and interpretation mechanism is used to identify, from the resulted lattices, all simple (1-to-1) and complex (N-to-1) substitutes for the failed operations in the defected service.

Although the approach is dedicated to the selection of service substitutes, it could be used at design time to select services from abstract service descriptions.

Finally, in chapter 7, we proposed a reputation management framework for web services and their providers. Reputation scores can be used as a reliability-indicator metric for selecting web services. Beside the architecture of the reputation managers, and the description of how users' feedback ratings are collected, managed and stored, we provided a reputation assessment model. The latter includes a penalization mechanism of suspicious users, to ensure an efficient reputation measurement from pure feedback ratings. The assessment model is built upon the distinction between positive and negative feedback ratings, with the inclusion of time sensitivity and user credibility factors, where the former is evaluated according to majority consensus. Moreover, we proposed a reputation bootstrapping mechanism to cope with the cold start and whitewashing problems. The mechanism allows a better introduction of newly published web services into the system, because efficient initial reputation values stabilize the performance of the whole system. We conducted a large experiment to investigate the feasibility and the practicability of the proposed model.

8.2 Future Work

The work presented in this thesis suggests various future research directions related to the orchestration of reliable web services. In this section, we outline some of these directions.

8.2.1 Improvement on the proposed approaches

- Firstly, we plan to automatically crawl, parse, index and compute similiarity between web services available in publication websites, such as ProgrammableWeb[1], WSindex[2], WebserviceX[3], WebserviceList[4] and serviceRepository [5]. The similarity scores between operations and services, with composability relationships are expected to be stored in relational databases. Thus, simple substitutes can be directly extracted using SQL commands. WSSIM have to integrate crawling capabilities, and have to manage storing and retrieving similarity scores.

- Secondly, we aim to study the similarity and composability relationships between stateful web services. In fact, the current work focuses on conventional Web services, which are stateless in nature as they use request and response messages for communication, without keeping any state between requests. However, some web services applications require services to record their communication. These stateful services require a precise interaction protocol for session management, which makes the investigation for substitution between such services a hard task.

- Thirdly, we plan to apply a machine learning algorithm for reputation and QoS prediction. Besides, we will focus on the study of trust and reputation of cloud services where significant challenges, due to the highly dynamic, distributed and non-transparent nature of cloud services, have to be addressed.

8.2.2 Formal Requirement Engineering Method

Many formal requirement engineering methods have been proposed in the domain of software engineering. Nevertheless, only few propositions could be applied in the context of Service-Oriented applications. Hence, there is a crucial need for

[1] www.programmableweb.com
[2] www.wsindex.org
[3] http://www.webservicex.net/ws/wscatlist.aspx
[4] www.webservicelist.com
[5] http://www.service-repository.com/

a formal requirement engineering method that assists developer to specify and document functional and non-functional requirements in web services orchestrations.

Such method allows the build of Use-case tools that support an automatic verification of desired qualities in different phases of orchestration's life-cycle.

8.2.3 Web services monitoring approach

One of the persistent challenges in service composition is quality awareness [128], which is achieved by service monitoring. Although, several methods have been proposed to standardize web service monitoring, none of these methods have been commonly adopted.

In addition, the available monitoring tools are dedicated for research purpose only, and they show their limitation with regards to the accuracy, overhead and scalability that is required by such tools [73]. Moreover, there is still a great demand for runtime-level quality verification. Therefore, runtime service monitoring with quality verification and adaptation is an open room for farther research and improvements, especially for providing dynamic lists of services with regular monitored QoS data updates.

Bibliography

[1] Ali Ait-Bachir. Measuring similarity of service interfaces. In *ICSOC PhD Symposium 2008*, page 59, 2008. (Cited on pages 57 and 59.)

[2] Eyhab Al-Masri and Qusay H Mahmoud. Qos-based discovery and ranking of web services. In *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, pages 529–534. IEEE, 2007. (Cited on page 152.)

[3] Zainab Mohammed Aljazzaf. *Trust-based service selection*. PhD thesis, The University of Western Ontario, 2011. (Cited on pages 38 and 41.)

[4] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, et al. Business process execution language for web services, 2003. (Cited on pages 21 and 23.)

[5] Danilo Ardagna, Luciano Baresi, Sara Comai, Marco Comuzzi, and Barbara Pernici. A service-based framework for flexible business processes. *IEEE software*, 28(2):61–67, 2011. (Cited on pages 44 and 47.)

[6] Ali Arsanjani. Service-oriented modeling and architecture. *IBM developer works*, pages 1–15, 2004. (Cited on page 73.)

[7] Lerina Aversano, Marcello Bruno, Gerardo Canfora, Massimiliano Di Penta, and Damiano Distante. Using concept lattices to support service selection. *International Journal of Web Services Research (IJWSR)*, 3(4):32–51, 2006. (Cited on pages 59 and 62.)

[8] Zeina Azmeh. *A Web service selection framework for an assisted SOA*. PhD thesis, Montpellier 2, 2011. (Cited on pages 18, 35, 36 and 38.)

[9] Zeina Azmeh, Maha Driss, Fady Hamoui, Marianne Huchard, Naouel Moha, and Chouki Tibermacine. Selection of composable web services driven by user requirements. In *IEEE International Conference on Web Services (ICWS)*, pages 395–402. IEEE, 2011. (Cited on pages 60 and 62.)

[10] Zeina Azmeh, Jean-Rémy Falleri, Marianne Huchard, and Chouki Tibermacine. Automatic web service tagging using machine learning and wordnet synsets. In *Web Information Systems and Technologies*, pages 46–59. Springer, 2011. (Cited on pages 61 and 138.)

[11] Zeina Azmeh, Marianne Huchard, Chouki Tibermacine, Christelle Urtado, and Sylvain Vauttier. Wspab: A tool for automatic classification & selection of web services using formal concept analysis. In *IEEE Sixth European Conference on Web Services(ECOWS'08).*, pages 31–40. IEEE, 2008. (Cited on pages 60 and 62.)

[12] Zeina Azmeh, Marianne Huchard, Chouki Tibermacine, Christelle Urtado, and Sylvain Vauttier. Using concept lattices to support web service compositions with backup services. In *Fifth International Conference on Internet and Web Applications and Services (ICIW)*, pages 363–368. IEEE, 2010. (Cited on pages 60 and 62.)

[13] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. (Cited on page 90.)

[14] Muneera Bano and Naveed Ikram. Issues and challenges of requirement engineering in service oriented software development. In *Software Engineering Advances (ICSEA), 2010 Fifth International Conference on*, pages 64–69. IEEE, 2010. (Cited on page 73.)

[15] Luciano Baresi and Sam Guinea. *Dynamo: Dynamic monitoring of WS-BPEL processes*, pages 478–483. Springer, 2005. (Cited on page 55.)

[16] Luciano Baresi, Sam Guinea, and Liliana Pasquale. Self-healing bpel processes with dynamo and the jboss rule engine. In *International workshop on Engineering of software services for pervasive environments: in conjunction with the 6th ESEC/FSE joint meeting*, pages 11–20. ACM, 2007. (Cited on page 55.)

[17] Márcio de Oliveira Barros and Arilo Claudio Dias Neto. Threats to validity in search-based software engineering empirical studies. techreport 0006/2011, UNIRIO - Universidade Federal do Estado do Rio de Janeiro, 2011. (Cited on pages 132 and 133.)

[18] Tom Bellwood, S Capell, L Clement, J Colgrave, MJ Dovey, D Feygin, AHR Kochman, P Macias, M Novotny, M Paolucci, et al. Universal description, discovery and integration specification (uddi) 3.0. *Online: http://uddi. org/pubs/uddi-v3. 00-published-20020719. htm*, 2005. (Cited on pages 2 and 17.)

[19] Boualem Benatallah, Marlon Dumas, and Quan Z Sheng. Facilitating the rapid development and scalable orchestration of composite web services.

Distributed and Parallel Databases, 17(1):5–37, 2005. (Cited on pages 2 and 20.)

[20] Daniela Berardi, Fahima Cheikh, Giuseppe De Giacomo, and Fabio Patrizi. Automatic service composition via simulation. *International Journal of Foundations of Computer Science*, 19(02):429–451, 2008. (Cited on page 44.)

[21] Domenico Bianculli, Walter Binder, Luigi Drago, and Carlo Ghezzi. Transparent reputation management for composite web services. In *Web Services, 2008. ICWS'08. IEEE International Conference on*, pages 621–628. IEEE, 2008. (Cited on pages 63 and 67.)

[22] Ben Bloch, Francisco Curbera, Y Goland, Neelakantan Kartha, CK Liu, S Thatte, and P Yendluri. Web services business process execution language. *OASIS Open Inc*, 2003. (Cited on page 21.)

[23] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970. (Cited on page 63.)

[24] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple object access protocol (soap) 1.1, 2000. (Cited on pages 2 and 17.)

[25] Francesco Buccafurri, Antonello Comi, Gianluca Lax, and Domenico Rosaci. A trust-based approach to clustering agents on the basis of their expertise. In *Agent and Multi-Agent Systems: Technologies and Applications*, pages 47–56. Springer, 2014. (Cited on page 63.)

[26] Jie Cao, Zhiang Wu, Youquan Wang, and Yi Zhuang. Hybrid collaborative filtering algorithm for bidirectional web service recommendation. *Knowledge and information systems*, 36(3):607–627, 2013. (Cited on page 162.)

[27] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Francesco Lo Presti, and Raffaela Mirandola. Qos-driven runtime adaptation of service oriented architectures. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 131–140. ACM, 2009. (Cited on page 51.)

[28] Fabio Casati and Ming-Chien Shan. Dynamic and adaptive composition of e-services. *Information systems*, 26(3):143–163, 2001. (Cited on pages 2, 20 and 44.)

[29] Sam Chapman, Barry Norton, and Fabio Ciravegna. Armadillo: Integrating knowledge for the semantic web. In *Proceedings of the Dagstuhl Seminar in Machine Learning for the Semantic Web*, February 2005. (Cited on page 90.)

[30] David A Chappell and Tyler Jewell. *Java web services*. Tecniche Nuove, 2002. (Cited on page 13.)

[31] Anis Charfi and Mira Mezini. Ao4bpel: An aspect-oriented extension to bpel. *World Wide Web*, 10(3):309–344, 2007. (Cited on page 44.)

[32] Harmeet Chawla, Haiping Xu, and MengChu Zhou. A real-time reliability model for ontology-based dynamic web service composition. In *SEKE*, pages 153–158, 2011. (Cited on page 52.)

[33] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. Web services description language (wsdl) version 2.0 part 1: Core language. World Wide Web Consortium, Recommendation REC-wsdl20-20070626, June 2007. (Cited on pages 2 and 16.)

[34] Jin-Hee Cho, Ananthram Swami, and Ray Chen. A survey on trust management for mobile ad hoc networks. *Communications Surveys & Tutorials, IEEE*, 13(4):562–583, 2011. (Cited on page 63.)

[35] Stéphani Chollet, Vincent Lestideau, Philippe Lalanda, Diana Moreno-Garcia, and Pierre Colomb. Heterogeneous service selection based on formal concept analysis. In *6th World Congress on Services (SERVICES-1)*, pages 367–374. IEEE, 2010. (Cited on pages 59 and 62.)

[36] William Cohen, Pradeep Ravikumar, and Stephen Fienberg. A comparison of string metrics for matching names and records. In *KDD Workshop on Data Cleaning and Object Consolidation*, volume 3, pages 73–78, 2003. (Cited on page 32.)

[37] Massimiliano Colombo, Elisabetta Di Nitto, and Marco Mauri. Scene: A service composition execution environment supporting dynamic changes disciplined through rules. In *Service-Oriented Computing–ICSOC 2006*, pages 191–202. Springer, 2006. (Cited on pages 44 and 48.)

[38] Antonello Comi, Lidia Fotia, Fabrizio Messina, Domenico Rosaci, and Giuseppe ML Sarnè. A qos-aware, trust-based aggregation model for grid federations. In *On the Move to Meaningful Internet Systems: OTM 2014 Conferences*, pages 277–294. Springer, 2014. (Cited on page 63.)

[39] William Conner, Arun Iyengar, Thomas Mikalsen, Isabelle Rouvellou, and Klara Nahrstedt. A trust management framework for service-oriented environments. In *Proceedings of the 18th international conference on World wide web*, pages 891–900. ACM, 2009. (Cited on pages 63 and 67.)

[40] Luca Console and WS-Diamond Team. Ws-diamond: An approach to web services-diagnosability, monitoring and diagnosis. In *International e-Challenges Conference, The Hague (October 2007)*, 2007. (Cited on page 55.)

[41] Vittorio Cortellessa and Vincenzo Grassi. Reliability modeling and analysis of service-oriented architectures. In *Test and analysis of web services*, pages 339–362. Springer, 2007. (Cited on page 50.)

[42] Marco Crasso, Alejandro Zunino, and Marcelo Campo. Query by example for web services. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 2376–2380. ACM, 2008. (Cited on pages 57 and 59.)

[43] Marco Crasso, Alejandro Zunino, and Marcelo Campo. A survey of approaches to web service discovery in service-oriented architectures. *Journal of Database Management (JDM)*, 22(1):102–132, 2011. (Cited on page 56.)

[44] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, Nirmal Mukhi, and S. Weerawarana. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *Internet Computing, IEEE*, 6(2):86–93, March 2002. (Cited on page 17.)

[45] Yu Dai, Lei Yang, and Bin Zhang. Qos-driven self-healing web service composition based on performance prediction. *Journal of Computer Science and Technology*, 24(2):250–261, 2009. (Cited on pages 18, 20 and 56.)

[46] Florian Daniel, Fabio Casati, Boualem Benatallah, and Ming-Chien Shan. Hosted universal composition: Models, languages and infrastructure in mashart. In *Conceptual Modeling-ER 2009*, pages 428–443. Springer, 2009. (Cited on page 44.)

[47] Valeria De Antonellis, Michele Melchiori, and Pierluigi Plebani. An approach to web service compatibility in cooperative processes. In *International Symposium on Applications and the Internet Workshops (SAINTW'06)*, pages 95–95. IEEE Computer Society, 2006. (Cited on pages 56 and 59.)

[48] Silvana De Gyvés Avila. *QoS awareness and adaptation in service composition*. PhD thesis, University of Leeds, 2014. (Cited on page 49.)

[49] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *journal of the american society for information scinece*, 41(6):391–407, 1990. (Cited on pages 29 and 90.)

[50] Alexander Dekhtyar. Lectures on knowledge discovery from data (course given at california polytechnic state university). `http://users.csc.calpoly.edu/~dekhtyar/466-Spring2012/`, 2012. Accessed: 2014-04-08. (Cited on page 30.)

[51] Demian Antony DŠMello, VS Ananthanarayana, and Supriya Salian. A review of dynamic web service composition techniques. In *Advanced Computing*, pages 85–97. Springer, 2011. (Cited on page 44.)

[52] Glen Dobson. Using ws-bpel to implement software fault tolerance for web services. In *32nd EUROMICRO Conference on Software Engineering and Advanced Applications, 2006. SEAA'06.*, pages 126–133. IEEE, 2006. (Cited on page 55.)

[53] Xin Dong, Alon Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang. Similarity search for web services. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 372–383. VLDB Endowment, 2004. (Cited on pages 56 and 59.)

[54] Maha Driss, Naouel Moha, Yassine Jamoussi, Jean-Marc Jézéquel, and Henda Hajjami Ben Ghézala. A requirement-centric approach to web service modeling, discovery, and selection. In *Service-Oriented Computing*, pages 258–272. Springer, 2010. (Cited on pages 60 and 62.)

[55] Jigyasu Dubey and V Tokekar. Bayesian network based trust model with time window for pure p2p computing systems. In *Wireless Computing and Networking (GCWCN), 2014 IEEE Global Conference on*, pages 219–223. IEEE, 2014. (Cited on page 41.)

[56] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. *International journal of web and grid services*, 1(1):1–30, 2005. (Cited on page 44.)

[57] Pascal Eck and Roel Wieringa. Requirements engineering for service-oriented computing: a position paper. 2003. (Cited on page 73.)

[58] Joyce El Hadad, Maude Manouvrier, and Marta Rukoz. Tqos: Transactional and qos-aware selection algorithm for automatic web service composition. *Services Computing, IEEE Transactions on*, 3(1):73–85, 2010. (Cited on pages 44 and 47.)

[59] Thomas Erl. *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 2006. (Cited on page 1.)

[60] Jean-Rémy Falleri, Zeina Azmeh, Marianne Huchard, Chouki Tibermacine, et al. Automatic tag identification in web service descriptions. In *WEBIST'10: The International Conference on Web Information Systems and Technology*, 2010. (Cited on page 138.)

[61] Xinxin Fan, Mingchu Li, Jianhua Ma, Yizhi Ren, Hui Zhao, and Zhiyuan Su. Behavior-based reputation management in p2p file-sharing networks. *Journal of Computer and System Sciences*, 78(6):1737–1750, 2012. (Cited on page 63.)

[62] Giuseppe Fenza and Sabrina Senatore. Friendly web services selection exploiting fuzzy formal concept analysis. *Soft Computing*, 14(8):811–819, 2010. (Cited on pages 60 and 62.)

[63] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000. (Cited on page 14.)

[64] Panayotis Fouliras. A novel reputation-based model for e-commerce. *Operational research*, pages 1–26, 2013. (Cited on page 63.)

[65] Maria Grazia Fugini and Enrico Mussi. Recovery of faulty web applications through service discovery. In *Proceedings of the 1st SMR-VLDB Workshop, Matchmaking and Approximate Semantic-based Retrieval: Issues and Perspectives, 32nd International Conference on Very Large Databases*, pages 67–80, 2006. (Cited on page 55.)

[66] Keita Fujii and Tatsuya Suda. Semantics-based context-aware dynamic service composition. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):12, 2009. (Cited on pages 44 and 47.)

[67] Bernhard Ganter, Gerd Stumme, and Rudolf Wille. *Formal Concept Analysis: foundations and applications*, volume 3626. springer, 2005. (Cited on page 33.)

[68] Martin Garriga, Andres Flores, Cristian Mateos, Alejandro Zunino, and Alejandra Cechich. Service selection based on a practical interface assessment scheme. *International Journal of Web and Grid Services*, 9(4):369–393, 2013. (Cited on pages 56, 57, 58, 59 and 130.)

[69] W3C Working Group et al. Qos for web services: requirements and possible approaches, 2003. (Cited on pages 18 and 20.)

[70] Ferry Hendrikx, Kris Bubendorfer, and Ryan Chard. Reputation systems: A survey and taxonomy. *Journal of Parallel and Distributed Computing*, 75:184–197, 2015. (Cited on page 63.)

[71] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004. (Cited on page 162.)

[72] San-Yih Hwang, Ee-Peng Lim, Chien-Hsiang Lee, and Cheng-Hung Chen. Dynamic web service selection for reliable web service composition. *Services Computing, IEEE Transactions on*, 1(2):104–116, 2008. (Cited on page 53.)

[73] Anne Immonen and Daniel Pakkala. A survey of methods and approaches for reliable dynamic service compositions. *Service Oriented Computing and Applications*, 3:17–24, 2014. (Cited on pages 2, 3, 50, 74, 158, 177 and 181.)

[74] Diana Inkpen. Csi4107: Information retrieval and the internet (course given at ottawa university, canada). `http://www.site.uottawa.ca/~diana/csi4107/`, 2015. Accessed: 2015-03-10. (Cited on pages 28 and 29.)

[75] Donovan Isherwood and Marijke Coetzee. Trust cv: Reputation-based trust for collectivist digital business ecosystems. In *Privacy, Security and Trust (PST), 2014 Twelfth Annual International Conference on*, pages 420–424. IEEE, 2014. (Cited on page 63.)

[76] Wassim Itani, Cesar Ghali, Ayman Kayssi, and Ali Chehab. Reputation as a service: A system for ranking service providers in cloud systems. In *Security, Privacy and Trust in Cloud Systems*, pages 375–406. Springer, 2014. (Cited on page 63.)

[77] Paul Jaccard. *Etude comparative de la distribution florale dans une portion des Alpes et du Jura*. Impr. Corbaz, 1901. (Cited on page 31.)

[78] M. A. Jaro. Probabilistic linkage of large public health data file. In *Statistics in Medicine*, volume 14, pages 491–498, 1995. (Cited on pages 32 and 90.)

[79] Audun Jøsang. Trust and reputation systems. In *Foundations of security analysis and design IV*, pages 209–245. Springer, 2007. (Cited on page 39.)

[80] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision support systems*, 43(2):618–644, 2007. (Cited on pages 39, 40 and 63.)

[81] Matjaz B Juric. A hands-on introduction to bpel. *Oracle (white paper)*, 2006. (Cited on page 21.)

[82] Swaroop Kalasapur, Mohan Kumar, and Behrooz Shirazi. Dynamic service composition in pervasive computing. *Parallel and Distributed Systems, IEEE Transactions on*, 18(7):907–918, 2007. (Cited on page 44.)

[83] Sravanthi Kalepu, Shonali Krishnaswamy, and Seng Wai Loke. Verity: a qos metric for selecting web services and providers. In *Web Information Systems Engineering Workshops, 2003. Proceedings. Fourth International Conference on*, pages 131–139. IEEE, 2003. (Cited on page 18.)

[84] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon, and Charlton Barreto. Web services choreography description language version 1.0. *W3C candidate recommendation*, 9, 2005. (Cited on pages 21 and 24.)

[85] Markus Keidl and Alfons Kemper. Towards context-aware adaptable web services. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 55–65. ACM, 2004. (Cited on pages 44 and 45.)

[86] Markus Keidl, Stefan Seltzsam, and Alfons Kemper. Flexible and reliable web service execution. In *Proc. of the 1st Workshop on Entwicklung von Anwendungen auf der Basis der XML Web-Service Technologie*, pages 17–30, 2002. (Cited on page 45.)

[87] Markus Keidl, Stefan Seltzsam, Konrad Stocker, and Alfons Kemper. Serviceglobe: distributing e-services across the internet. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 1047–1050. VLDB Endowment, 2002. (Cited on page 45.)

[88] Natallia Kokash. A comparison of web service interface similarity measures. *Frontiers in Artificial Intelligence and Applications*, 142:220, 2006. (Cited on pages 56, 57, 58 and 59.)

[89] Kyriakos Kritikos and Dimitris Plexousakis. Requirements for qos-based web service description and discovery. *Services Computing, IEEE Transactions on*, 2(4):320–337, 2009. (Cited on page 73.)

[90] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955. (Cited on page 96.)

[91] Freddy Lécué, Yosu Gorronogoitia, Rafael Gonzalez, Mateusz Radzimski, and Matteo Villa. Soa4all: An innovative integrated approach to services composition. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 58–67. IEEE, 2010. (Cited on pages 44 and 46.)

[92] VI Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707, 1966. (Cited on pages 32 and 90.)

[93] Yingmin Li, Tarek Melliti, and Philippe Dague. Modeling BPel web services for diagnosis: Towards self-healing web services. In *WEBIST (1)*, pages 297–304, 2007. (Cited on page 5.)

[94] Noura Limam and Raouf Boutaba. Assessing software service quality and trustworthiness at selection time. *IEEE Transactions on Software Engineering,*, 36(4):559–574, 2010. (Cited on pages 64, 67 and 154.)

[95] Xin Liu, Anwitaman Datta, and Krzysztof Rzadca. Trust beyond reputation: A computational trust model based on stereotypes. *Electronic Commerce Research and Applications*, 12(1):24–39, 2013. (Cited on page 63.)

[96] Miodrag Lovric. *International encyclopedia of statistical science*. Springer London, 2011. (Cited on page 26.)

[97] Michael R Lyu et al. *Handbook of software reliability engineering*, volume 222. IEEE computer society press CA, 1996. (Cited on page 2.)

[98] Jiang Ma and Hao-peng Chen. A reliability evaluation framework on composite web service. In *Service-Oriented System Engineering, 2008. SOSE'08. IEEE International Symposium on*, pages 123–128. IEEE, 2008. (Cited on page 54.)

[99] Ismat Maarouf, Uthman Baroudi, and Abdurahim R Naseer. Efficient monitoring approach for reputation system-based trust-aware routing in wireless sensor networks. *IET communications*, 3(5):846–858, 2009. (Cited on page 157.)

[100] Ross A Malaga. Web-based reputation management systems: Problems and suggested solutions. *Electronic Commerce Research*, 1(4):403–417, 2001. (Cited on page 41.)

[101] Zaki Malik and Athman Bouguettaya. Rateweb: Reputation assessment for trust establishment among web services. *The VLDB JournalâĂŤThe International Journal on Very Large Data Bases*, 18(4):885–911, 2009. (Cited on pages 64, 67, 139, 153, 157, 158, 160 and 165.)

[102] Zaki Malik and Athman Bouguettaya. Reputation bootstrapping for trust establishment among web services. *Internet Computing, IEEE*, 13(1):40–47, 2009. (Cited on pages 67, 153 and 157.)

[103] Félix Gómez Mármol and Marcus Quintino Kuhnen. Reputation-based web service orchestration in cloud computing: A survey. *Concurrency and Computation: Practice and Experience*, 2013. (Cited on pages 63, 67, 68 and 143.)

[104] Andrea Maurino, Enrico Mussi, Stefano Modafferi, and Barbara Pernici. The MAIS framework for composite web services. *IBIS*, 6:32–64, 2007. (Cited on page 44.)

[105] E Michael Maximilien and Munindar P Singh. Conceptual model of web service reputation. *Acm Sigmod Record*, 31(4):36–41, 2002. (Cited on page 6.)

[106] Brahim Medjahed and Athman Bouguettaya. A multilevel composability model for semantic web services. *Knowledge and Data Engineering, IEEE Transactions on*, 17(7):954–968, 2005. (Cited on pages 44 and 45.)

[107] Loubna Mekouar and Youssef Iraqi. Trustws: A trust management system for web services. In *International Symposium on Web Services, At Dubai, UAE*, 2010. (Cited on pages 65 and 170.)

[108] Loubna Mekouar, Youssef Iraqi, and Raouf Boutaba. Incorporating trust in network virtualization. In *2010 IEEE 10th International Conference on Computer and Information Technology (CIT)*, pages 942–947. IEEE, 2010. (Cited on pages 65, 67 and 139.)

[109] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, pages 117–, Washington, DC, USA, 2002. IEEE Computer Society. (Cited on pages 87, 92 and 97.)

[110] Anton Michlmayr, Florian Rosenberg, Philipp Leitner, and Schahram Dustdar. End-to-end support for qos-aware service selection, binding, and mediation in vresco. *Services Computing, IEEE Transactions on*, 3(3):193–205, 2010. (Cited on pages 44 and 47.)

[111] Nikola Milanovic and Miroslaw Malek. Current solutions for web service composition. *IEEE Internet Computing*, 8(6):51–59, 2004. (Cited on page 44.)

[112] C Mohan. Dynamic e-business: Trends in web services. In *Technologies for E-Services*, pages 1–5. Springer, 2002. (Cited on page 13.)

[113] Shahab Mokarizadeh, Nima Dokoohaki, Mihhail Matskin, and Peep Küngas. Trust and privacy enabled service composition using social experience. In *Software Services for e-World*, pages 226–236. Springer, 2010. (Cited on pages 64 and 67.)

[114] Oliver Moser, Florian Rosenberg, and Schahram Dustdar. Non-intrusive monitoring and service adaptation for ws-bpel. In *Proceedings of the 17th international conference on World Wide Web*, pages 815–824. ACM, 2008. (Cited on pages 44 and 48.)

[115] Hamid Reza Motahari Nezhad, Boualem Benatallah, Axel Martens, Francisco Curbera, and Fabio Casati. Semi-automated adaptation of service interactions. In *Proceedings of the 16th international conference on World Wide Web*, pages 993–1002. ACM, 2007. (Cited on page 131.)

[116] Lik Mui, Mojdeh Mohtashemi, Cheewee Ang, Peter Szolovits, and Ari Halberstadt. Ratings in distributed systems: A bayesian approach. (Cited on page 41.)

[117] Lik Mui, Mojdeh Mohtashemi, and Ari Halberstadt. A computational model of trust and reputation. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 2431–2439. IEEE, 2002. (Cited on page 41.)

[118] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970. (Cited on page 90.)

[119] Anne HH Ngu, Michael Pierre Carlson, Quan Z Sheng, and Hye-young Paik. Semantic-based mashup of composite applications. *Services Computing, IEEE Transactions on*, 3(1):2–15, 2010. (Cited on page 44.)

[120] Hien Trang Nguyen, Jian Yang, and Weiliang Zhao. Bootstrapping trust and reputation for web services. In *Commerce and Enterprise Computing (CEC), 2012 IEEE 14th International Conference on*, pages 41–48. IEEE, 2012. (Cited on page 153.)

[121] Hien Trang Nguyen, Weiliang Zhao, and Jian Yang. A trust and reputation model based on bayesian network for web services. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 251–258. IEEE, 2010. (Cited on pages 6 and 67.)

[122] Talal H Noor, Quan Z Sheng, Sherali Zeadally, and Jian Yu. Trust management of services in cloud environments: Obstacles and solutions. *ACM Computing Surveys (CSUR)*, 46(1):12, 2013. (Cited on page 63.)

[123] Bart Orriens and Jian Yang. A rule driven approach for developing adaptive service oriented business collaboration. In *Services Computing, 2006. SCC'06. IEEE International Conference on*, pages 182–189. IEEE, 2006. (Cited on pages 44 and 46.)

[124] Meriem Ouederni, Gwen Salaün, and Ernesto Pimentel. Measuring the compatibility of service interaction protocols. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 1560–1567, New York, NY, USA, 2011. ACM. (Cited on pages 57 and 59.)

[125] Michael Papazoglou. *Web services: principles and technology*. Pearson Education, 2008. (Cited on pages 14, 17, 18 and 20.)

[126] Michael P Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: a research roadmap. *International Journal of Cooperative Information Systems*, 17(02):223–255, 2008. (Cited on page 49.)

[127] Mike P Papazoglou and Willem-Jan Van Den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB journal*, 16(3):389–415, 2007. (Cited on page 1.)

[128] MP Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007. (Cited on pages 1, 49 and 181.)

[129] Chris Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003. (Cited on page 20.)

[130] Dunlu Peng and Qingkui Chen. An efficient approach for managing replaceability of web services. In *Proceedings of the 2008 Fourth International Conference on Semantics, Knowledge and Grid*, SKG '08, pages 388–391, Washington, DC, USA, 2008. IEEE Computer Society. (Cited on pages 59 and 62.)

[131] Isaac Pinyol and Jordi Sabater-Mir. Computational trust and reputation models for open multi-agent systems: a review. *Artificial Intelligence Review*, 40(1):1–25, 2013. (Cited on page 63.)

[132] Giuseppe Pirró. A semantic similarity metric combining features and intrinsic information content. *Data Knowl. Eng.*, 68:1289–1308, November 2009. (Cited on pages 90 and 97.)

[133] Pierluigi Plebani and Barbara Pernici. Urbe: Web service retrieval based on similarity evaluation. *IEEE Trans. on Knowl. and Data Eng.*, 21:1629–1642, November 2009. (Cited on pages ix, 57, 58, 59, 94 and 95.)

[134] Edy Portmann, Andreas Meier, Philippe Cudré-Mauroux, and Witold Pedrycz. Fora–a fuzzy set based framework for online reputation management. *Fuzzy Sets and Systems*, 2014. (Cited on page 41.)

[135] Edy Portmann and Witold Pedrycz. Fuzzy web knowledge aggregation, representation, and reasoning for online privacy and reputation management. In *Fuzzy Cognitive Maps for Applied Sciences and Engineering*, pages 89–105. Springer, 2014. (Cited on page 41.)

[136] Jinghai Rao and Xiaomeng Su. *A survey of automated web service composition methods*, pages 43–54. Springer, 2005. (Cited on page 44.)

[137] M Tamer Refaei, Luiz A DaSilva, Mohamed Eltoweissy, and Tamer Nadeem. Adaptation of reputation management systems to dynamic network conditions in ad hoc networks. *Computers, IEEE Transactions on*, 59(5):707–719, 2010. (Cited on page 63.)

[138] Paul Resnick, Richard Zeckhauser, John Swanson, and Kate Lockwood. The value of reputation on ebay: A controlled experiment. *Experimental Economics*, 9(2):79–101, 2006. (Cited on page 41.)

[139] Juan Manuel Rodriguez, Marco Crasso, Alejandro Zunino, and Marcelo Campo. Improving web service descriptions for effective service discovery. *Science of Computer Programming*, 75(11):1001–1021, 2010. (Cited on page 130.)

[140] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975. (Cited on page 27.)

[141] A Lazcano G Alonso H Schuldt and C Schuler. The wise approach to electronic commerce. *Int'l J. Computer Systems Science and Eng*, 15(5):343–355, 2000. (Cited on page 44.)

[142] Toby Segaran. *Programming collective intelligence: building smart web 2.0 applications*. " O'Reilly Media, Inc.", 2007. (Cited on page 30.)

[143] Ali Shaikh Ali, Shalil Majithia, Omer F Rana, and David W Walker. Reputation-based semantic service discovery. *Concurrency and Computation: Practice and Experience*, 18(8):817–826, 2006. (Cited on page 6.)

[144] Quan Z Sheng, Boualem Benatallah, Marlon Dumas, and Eileen Oi-Yan Mak. Self-serv: a platform for rapid composition of web services in a peer-to-peer environment. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 1051–1054. VLDB Endowment, 2002. (Cited on pages 20, 44 and 45.)

[145] Quan Z Sheng, Boualem Benatallah, Zakaria Maamar, and Anne HH Ngu. Configurable composition and adaptive provisioning of web services. *Services Computing, IEEE Transactions on*, 2(1):34–49, 2009. (Cited on page 49.)

[146] Quan Z Sheng, Xiaoqiang Qiao, Athanasios V Vasilakos, Claudia Szabo, Scott Bourne, and Xiaofei Xu. Web services composition: A decade's overview. *Information Sciences*, 280:218–238, 2014. (Cited on pages vii, 2, 20, 21, 22, 23, 24, 25, 26, 44 and 49.)

[147] Jocelyn Simmonds, Shoham Ben-David, and Marsha Chechik. Guided recovery for web service applications. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 247–256. ACM, 2010. (Cited on page 55.)

[148] Jocelyn Simmonds, Shoham Ben-David, and Marsha Chechik. *Monitoring and recovery of web service applications*, pages 250–288. Springer, 2010. (Cited on page 55.)

[149] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. In *Journal of Molecular Biology*, volume 147(1), pages 195–197, 1981. (Cited on page 90.)

[150] Karen Spärck Jones. Idf term weighting and ir research lessons. *Journal of Documentation*, 60(5):521–523, 2004. (Cited on page 27.)

[151] Giorgos Stoilos, Giorgos Stamou, and Stefanos Kollias. A string metric for ontology alignment. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *Proceedings of the 4rd International Semantic Web Conference (ISWC)*, pages 624–637, Berlin, Heidelberg, November 2005. Springer. (Cited on page 90.)

[152] Eleni Stroulia and Yiqiao Wang. Structural and semantic matching for assessing web-service similarity. *International Journal of Cooperative Information Systems*, 14:407–437, 2005. (Cited on page 94.)

[153] K Adlin Suji and S Sujatha. A comprehensive survey of web service choreography, orchestration and workflow building. *International Journal of Computer Applications*, 88(13):18–23, 2014. (Cited on page 44.)

[154] Rajesh Sumra and D Arulazi. Quality of service for web services-demystification, limitations, and best practices. *Retrieved February*, 10:2006, 2003. (Cited on pages 18 and 20.)

[155] Hua Sun, Jiong Yu, Zhen Yu Zhang, Li Li, and Bin Liao. Evaluation of trustworthiness based on fuzzy set theory. *International Journal of Future Generation Communication & Networking*, 7(2), 2014. (Cited on page 41.)

[156] Girish Suryanarayana and Richard N Taylor. A survey of trust management and resource discovery technologies in peer-to-peer applications. 2004. (Cited on page 40.)

[157] Suhas Sutariya and Prashant Modi. A review of different reputation schemes to thwart the misbehaving nodes in mobile ad hoc network. *International Journal of Computer Science & Information Technologies*, 5(3), 2014. (Cited on page 63.)

[158] Yang Syu, Shang-Pin Ma, Jong-Yin Kuo, and Yong-Yi FanJiang. A survey on automated service composition methods and related techniques. In *Services Computing (SCC), 2012 IEEE Ninth International Conference on*, pages 290–297. IEEE, 2012. (Cited on page 44.)

[159] Steven Tadelis. Firm reputation with hidden information. *Economic Theory*, 21(2-3):635–651, 2003. (Cited on page 40.)

[160] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005. (Cited on page 31.)

[161] Karim Tari, Yacine Amirat, Abdelghani Chibani, Ali Yachir, and Abdelhamid Mellouk. Context-aware dynamic service composition in ubiquitous environment. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1–6. IEEE, 2010. (Cited on page 50.)

[162] Maurice Ter Beek, Antonio Bucchiarone, and Stefania Gnesi. Web service composition approaches: From industrial standards to formal methods. In *Internet and Web Applications and Services, 2007. ICIW'07. Second International Conference on*, pages 15–15. IEEE, 2007. (Cited on page 44.)

[163] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Fourth Edition*. Academic Press, 4th edition, 2008. (Cited on page 30.)

[164] Okba Tibermacine, Chouki Tibermacine, and Foudil Cherif. Wssim: a tool for the measurement of web service interface similarity. In *French-speaking Conference on Software Architectures (CAL'13)*, 2013. (Cited on page 7.)

[165] Okba Tibermacine, Chouki Tibermacine, and Foudil Cherif. A practical approach to the measurement of similarity between wsdl-based web services. *Revue des Nouvelles Technologies de l'Information*, 6th French-speaking Conference on Software Architectures, RNTI-L-7:03–18, 2014. (Cited on pages 7, 138, 146 and 149.)

[166] Okba Tibermacine, Chouki Tibermacine, and Foudil Cherif. A process to identify relevant substitutes for healing failed ws-* orchestrations. *Journal of Systems and Software*, 104(0):1 – 16, 2015. (Cited on page 7.)

[167] Okba Tibermacine, Chouki Tibermacine, and Foudil Cherif. Regression-based bootstrapping of web service reputation measurement. In *to appear in Proceedings of the 13th IEEE International Conference on Web Services (ICWS'15), Application Track*. IEEE Computer Society, June July 2015. (Cited on page 8.)

[168] Hong-Linh Truong, Robert Samborski, and Thomas Fahringer. Towards a framework for monitoring and analyzing qos metrics of grid services. In *e-Science and Grid Computing, 2006. e-Science'06. Second IEEE International Conference on*, pages 65–65. IEEE, 2006. (Cited on pages vii, 18 and 19.)

[169] Wei-Tek Tsai. Service-oriented system engineering: a new paradigm. In *Service-oriented system engineering, 2005. sose 2005. IEEE International Workshop*, pages 3–6. IEEE, 2005. (Cited on page 73.)

[170] W.T. Tsai, Z. Jin, P. Wang, and B. Wu. Requirement engineering in service-oriented system engineering. In *e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on*, pages 661–668, Oct 2007. (Cited on page 73.)

[171] Esko Ukkonen. Approximate string-matching with q-grams and maximal matches. In *Theoretical Computer Science*, volume 92, pages 191–211, 1992. (Cited on page 90.)

[172] VRSD Vijayakumar, RSD Wahida Banu, and Jemal H Abawajy. An efficient approach based on trust and reputation for secured selection of grid resources. *International journal of parallel, emergent and distributed systems*, 27(1):1–17, 2012. (Cited on page 63.)

[173] Le-Hung Vu, Manfred Hauswirth, and Karl Aberer. Qos-based service selection and ranking with trust and reputation management. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, pages 466–483. Springer, 2005. (Cited on page 6.)

[174] Web Services Architecture Working Group W3C et al. Web services architecture requirements. *W3C Working Draft*, 2002. (Cited on page 14.)

[175] Lijun Wang, Xiaoying Bai, Lizhu Zhou, and Yinong Chen. A hierarchical reliability model of service-based software system. In *Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International*, volume 1, pages 199–208. IEEE, 2009. (Cited on page 52.)

[176] S. Wang, Z. Zheng, Z. Wu, M. Lyu, and F. Yang. Reputation measurement and malicious feedback rating prevention in web service recommendation systems. *IEEE Transactions on Services Computing*, PP(99):1–1, 2014. (Cited on pages 6, 63, 65, 66, 67, 157, 169 and 170.)

[177] Yao Wang and Julita Vassileva. Toward trust and reputation based web service selection: A survey. *International Transactions on Systems Science and Applications*, 3(2):118–132, 2007. (Cited on pages 6 and 63.)

[178] Andrew Whitby, Audun Jøsang, and Jadwiga Indulska. Filtering out unfair ratings in bayesian reputation systems. In *Proc. 7th Int. Workshop on Trust in Agent Societies*, volume 6, 2004. (Cited on pages 165 and 168.)

[179] Stephen A White et al. Business process modeling notation. *Specification, BPMI. org*, 2004. (Cited on page 21.)

[180] Rudolf Wille. *Restructuring lattice theory: an approach based on hierarchies of concepts*. Springer, 2009. (Cited on page 33.)

[181] William E. Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. In *Proceedings of the Section on Survey Research*, pages 354–359, 1990. (Cited on pages 32 and 90.)

[182] Martin Wirsing and Matthias Hölzl. *Rigorous Software Engineering for Service-oriented Systems: Results of the SENSORIA Project on Software Engineering for Service-oriented Computing*, volume 6582. Springer Science & Business Media, 2011. (Cited on page 73.)

[183] Ryan Wishart, Ricky Robinson, Jadwiga Indulska, and Audun Jøsang. Superstringrep: reputation-enhanced service discovery. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume*

38, pages 49–57. Australian Computer Society, Inc., 2005. (Cited on pages 6 and 140.)

[184] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999. (Cited on page 90.)

[185] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Bjöorn Regnell, and Anders Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000. (Cited on page 132.)

[186] Li Xiong and Ling Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Transactions on Knowledge and Data Engineering,*, 16(7):843–857, 2004. (Cited on page 157.)

[187] Ziqiang Xu, Patrick Martin, Wendy Powley, and Farhana Zulkernine. Reputation-enhanced qos-based web services discovery. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 249–256. IEEE, 2007. (Cited on pages 139 and 140.)

[188] Xinfeng Ye, Jupeng Zheng, and Bakh Khoussainov. A robust service recommendation scheme. In *Services Computing (SCC), 2013 IEEE International Conference on*, pages 73–80. IEEE, 2013. (Cited on page 6.)

[189] Serhiy A Yevtushenko. System of data analysis Şconcept explorerŤ. In *Proceedings of the 7th national conference on Artificial Intelligence KII*, volume 2000, 2000. (Cited on page 38.)

[190] Serhiy A Yevtushenko. System of data analysis "concept explorer". In *Proceedings of the 7th national conference on Artificial Intelligence KII*, volume 2000, 2000. (Cited on page 123.)

[191] Jian Yu, Jun Han, Quan Z Sheng, and Steven O Gunarso. Percas: an approach to enabling dynamic and personalized adaptation for context-aware services. In *Service-Oriented Computing*, pages 173–190. Springer, 2012. (Cited on page 44.)

[192] Qi Yu and Athman Bouguettaya. *Foundations for efficient web service selection*. Springer Science & Business Media, 2009. (Cited on pages 14 and 20.)

[193] Alireza Zarghami, Soude Fazeli, Nima Dokoohaki, and Mihhail Matskin. Social trust-aware recommendation system: A t-index approach. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on*

Web Intelligence and Intelligent Agent Technology-Volume 03, pages 85–90. IEEE Computer Society, 2009. (Cited on page 64.)

[194] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z Sheng. Quality driven web services composition. In *Proceedings of the 12th international conference on World Wide Web*, pages 411–421. ACM, 2003. (Cited on page 45.)

[195] Liangzhao Zeng, Boualem Benatallah, Anne HH Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *Software Engineering, IEEE Transactions on*, 30(5):311–327, 2004. (Cited on page 53.)

[196] Wen Zhang, Taketoshi Yoshida, and Xijin Tang. Tfidf, lsi and multi-word in information retrieval and text categorization. In *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on*, pages 108–113. IEEE, 2008. (Cited on page 27.)

[197] Yilei Zhang, Zibin Zheng, and Michael R Lyu. Wspred: A time-aware personalized qos prediction framework for web services. In *IEEE 22nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 210–219. IEEE, 2011. (Cited on page 127.)

[198] Huanyu Zhao and Xiaolin Li. Vectortrust: trust vector aggregation scheme for trust management in peer-to-peer networks. *The Journal of Supercomputing*, 64(3):805–829, 2013. (Cited on page 63.)

[199] Zibin Zheng and Michael R Lyu. Personalized reliability prediction of web services. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 22(2):12, 2013. (Cited on page 49.)

[200] Zibin Zheng, Yilei Zhang, and Michael R Lyu. Distributed qos evaluation for real-world web services. In *Proc. of ICWS'10*, pages 83–90. IEEE, 2010. (Cited on page 152.)

[201] Runfang Zhou and Kai Hwang. Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing. *IEEE Transactions on Parallel and Distributed Systems*, 18(4):460–473, 2007. (Cited on page 63.)

[202] Alejandro Zunino and Marcelo Campo. A survey of approaches to web service discovery in serviceŮoriented architectures. *Innovations in Database Design, Web Applications, and Information Systems Management*, page 107, 2012. (Cited on page 44.)