# Trustworthiness in Service Oriented Computing

Nicola Miotto

Kongens Lyngby 2011
IMM-PHD-2011-70

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

# Summary

Service Oriented Computing is an emerging paradigm for distributed computing, where Web Services represent the bricks of Service Oriented Architecture. Brought to its full potential, this vision could allow software developers to take advantage of agents to automatically discover and compose Web Services over the Internet to build a distributed system. In the past years, there have been many issues discussed about web services, regarding their implementation, their founding principles and so on. But there is still a concern that would need to be investigated: the trustworthiness provisioning.
The aim of this thesis is to provide a complete study about this issue in Service Oriented Computing environments.

In Chapter 1 we aim at explaining the rationale behind SOA, discussing about the historical reasons that gave life to this paradigm. Then, an explanation of the current definition of *Service Oriented Paradigm* will be provided, showing different use cases involving this concept. We will illustrate how different technologies try to meet the requirements of this pattern, focusing the attention on those that nowadays are considered the de facto standard to concretise the *service orientation vision.*

Essentially, in Chapter 2, an exhaustive analysis of the state of the art concerning *trust provisioning* is provided. Trust provisioning is discussed both broadly speaking and refining more details specifically related to the Service Oriented Computing environment.
The first contribution of this work has been to categorize the most notable works in few categories, depending on the rationale of their approach. This way it has been possible to highlight the shortcomings and the advantages of each category.

The whole study is accompanied by a running example illustrating a possible real-world scenario for each approach. This should help the reader to better understand the discussed issues.
The final part of the analysis further sum-up the issues arisen during the first part, focusing the attention on the reasons that may determine the effectiveness of a possible new approach.

The discussion carried out in Chapter 2 has been the preliminary study that let us work on the second contribution of this work: the definition of the founding principles that should be taken into consideration while designing a new framework for trust provisioning in SOA. These principles have been outlined in light of the state of the art analysis and supported by the drawn considerations and conclusions.
Based on these principles a new framework has been suggested. Considering the vast amount of studies and frameworks already proposed in literature, the effort has been to devise a new solution adopting, where possible, already defined and tested solutions.
The framework is described in Chapter 3 and it is the result of the studies conducted in the first part. It constitutes the third contribution of this thesis.
As mentioned afore, the solution has been composed by joining different works discussed in literature. Some of them has been extended and adapted to fit our new framework, solving some of the major problems listed in Chapter 2. All those interventions on the existing approaches are further contributions that this project presents.

In Chapter 4, the results are discussed to provide practical credibility to our work. Considering no implemented solutions have been provided (for time limitations), we tried to explain by means of some real-world test cases how the framework should behave and solve the arisen problems. Some implementative suggestions are also provided in order to show how the framework could possibly be concretised in a working solution.

Finally, in chapter 5, an argumentation about how the issues has been solved is provided, along with a discussion regarding the future work that could improve the suggested framework. Furthermore, the contributions of the thesis are explained in more detail, contextualizing them to the work illustrated throughout the document.

# Papers included in the thesis

[A] Nicola Dragoni, Nicola Miotto, Davide Papini Analysis of Trust-Based Approaches for Web Service Selection. *5th Nordic Workshop on Dependability and Security (NODES11)*, 2011. Accepted for publication.

# Acknowledgements

I wish to thank my supervisor Prof. Nicola Dragoni for having welcomed me at DTU to carry out my master thesis and for the effort in helping me reaching a valuable result.
I also thank Prof. Claudio Palazzi, my relator in Italy, for having helped me, when possible, to improve my thesis.
Thanks to Davide Papini for the initial help in finding contacts and suggestions to start my thesis at DTU. And thanks for having presented my paper at the NODES11 workshop and for having introduced me the people of the IMM department.
Thanks to both Fontas Fafoutis and Gianluca Frison for having kindly hosted me during my last period in Denmark.
Thanks to my girlfriend Laura for having put up with me during my anxiety moments.
A final great thanks to my parents and my grandparents that supported my stay in Denmark.

# Contents

**Summary** **i**

**Papers included in the thesis** **iii**

**Acknowledgements** **v**

**1 Background** **1**
- 1.1 Genesis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
- 1.2 Issues of a short-term-benefit IT solutions . . . . . . . . . . . . 2
- 1.3 The SOA vision . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
- 1.4 SOA Technologies . . . . . . . . . . . . . . . . . . . . . . . . . . 8
- 1.5 Use cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
- 1.6 Trust in SOA . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

**2 Trust in SOA - State of the Art** **17**
- 2.1 Terms confusion . . . . . . . . . . . . . . . . . . . . . . . . . . 17
- 2.2 Sources of trustworthiness . . . . . . . . . . . . . . . . . . . . . 18
- 2.3 Suggested approaches . . . . . . . . . . . . . . . . . . . . . . . 19

**3 Framework** **51**
- 3.1 Founding principles . . . . . . . . . . . . . . . . . . . . . . . . . 52
- 3.2 High Level Architecture . . . . . . . . . . . . . . . . . . . . . . 55
- 3.3 Evidence supplier . . . . . . . . . . . . . . . . . . . . . . . . . . 59
- 3.4 QoS based step . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
- 3.5 Reputation based selection step . . . . . . . . . . . . . . . . . . 74
- 3.6 Agreement negotiation and monitoring . . . . . . . . . . . . . . 82

**4 Application** **91**
4.1 Test cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91
4.2 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 101

**5 Conclusions** **107**
5.1 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . 108
5.2 Future Works . . . . . . . . . . . . . . . . . . . . . . . . . . . 110

**A Publication - NODES11** **113**

Chapter 1

# Background

Throughout the first chapter of this thesis, we will describe the background of the Service Orientation paradigm.
Firstly, we will illustrate part of the history of this vision, in order to highlight the reasons and the issues the made this new concept born. Essentially, we will describe what Service Oriented Computing is, illustrating the definitions and the ancestors of this paradigm. Furthermore, the technologies that have been devised to carry out the rationale of this vision are briefly illustrated, focusing on those that nowadays are considered the de facto standard for SOC. Finally, the issue of *trustworthiness* provisioning is mentioned, in order to provide an idea of what will be the main topic of discussion throughout the rest of the document.

## 1.1 Genesis

In order to clearly understand what a Service Oriented Architecture is and why it is gaining such a great attention in the recent years, it is necessary to jump back to several years ago searching for the roots of this architectural pattern.
After the beginning of the information technology revolution (during the '70s), a new era started for the world of information. Lots of people could afford their

own computer, but also lots of companies started to foresee the economical advantages this revolution may have carried out.
The world of business could take a huge benefit from automating the business tasks and from delegating the internal business logic to a bunch of software solutions. And the IT agencies began to spread out all over the world, offering automated systems for the typical enterprise-level problems, like customer relationship management, document management, human resource management and so on. It is the *Electronic Business*, better known as *e-business*, era. This word appeared the first time in an article published in *Macworld Communications Inc*[35] and eventually spread out in October, 1997, when IBM launched a thematic campaign built around the term. In the following years an increasing amount of companies started to use the Web to buy components and supplies from other companies, to collaborate on sales promotions, and to carry out joint research. It was then necessary to take advantage of the Internet to improve the communication of information among enterprises. That was one of the main reasons why concepts like *interoperability*, *reusability*, *abstraction* began to get a critical importance for the economical perspective of each company. This made the enterprises face a critical problem: each company's business logic and data were tied to the company specific protocols, technologies, languages and so on.

When it came to share data and services over the Internet between enterprises, an accepted and popular approach was to design and develop ad-hoc solutions for each need. This way each time it came to integrate many different components or to reuse already existing software, many problems were rising. That is why the idea of *service oriented architecture* started to take place.

## 1.2 Issues of a short-term-benefit IT solutions

As we mentioned above, there are many issues that might emerge at the enterprise level when dealing with automation of business tasks. The most common and most accepted approaches were those providing a relatively quick solution and an immediate tangible result. An approach that in problem solving theory would be defined as *greedy*: a locally optimal choice at each stage, often leading to a globally sub-optimal solution. Let us provide a concrete scenario to better explain the problem:

- Encom is a big enterprise working on the IT field;
- the company realizes that it is necessary to provide itself with an internal employee management system;

- after a first analysis of the business problem, the hired IT specialist designs a solution capable of an authentication, authorization, logging and employee profile management system;
- to better exploit the company know-how, the system is eventually developed with the Java programming language.
- After few months, the security branch of the Encom Enterprise asks for integrating the current monitoring system with a remote interface, provided with an authentication/authorization system. This was necessary because the security managers of the company would have been transferred abroad in 2 months;
- The IT specialist is then hired again and asked for developing such a kind of platform, paying attention to the backward-compatibility issues related to the language used for the currently existing system: C++.
- The quickest and cheapest way the IT specialist can take is to reimplement the authentication/authorization system for the current problem using the C++ language.

The example lacks of several details, but it can quite clearly focus the problem. The solution the IT specialist provided in the first case was of course the one with the most short-term tangible benefit. But it was not designed to be eventually reused or integrated. The same authentication/authorization system could have been used also for the next required solution.
Spreading the problem out of the boundaries of the same enterprise, there are many other similar scenarios we can describe:

- A second company, let us call it Sirius Cybernetics, has not enough developers available for starting up a project, consisting of a logging service for the employee activities in the internal management system. It needs then to outsource it.
- Encom is asked for developing the system. Another IT developer is then hired in order to carry on the solution, the same way as described in the previous example: a new team is instantiated to integrate a brand new logging tool into the Sirius Cybernetics system. The operation takes months, resulting in an expensive product and in a waste of time for both companies.

First of all, as said before, the Encom Enterprise may have taken advantage of its already developed logging system to reuse it for the new solution. But it was put in a monolithic application, making it difficult to export it for reuse.

Moreover, the Sirius Cybernetics enterprise could have searched for the already implemented solution and, after having discovered it, it may have integrated it as an additional distributed component to its system, without introducing new code to maintain in its current infrastructure.
So, how to fulfill the "Don't Repeat Yourself" principle? The idea is that for each complex problem, many small concerns should be identified. Each concern should be solved as a small logic unit, with a standard interface and a contract as well, in order to allow its reuse and intercommunication with other units. These small unities are named, in the context we are talking about, *services.*

## 1.3 The SOA vision

### 1.3.1 The Bricks: Services

The OASIS[1] definition of the term *service* is:

> *A mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.*[64]

In other terms, a *service* is a collection of functionally related capabilities. Each service displays an interface that plays the role of a contract. An interface discloses a set of operations that the service provides for. More services together can be orchestrated to achieve e more complex goal. Let us provide a really high level example:

- A company needs to design a *vacation planning* system architecture for a tourism agency;
- after a thorough analysis, they decide to break up the problem into different components:
  - Trip planning
  - Accommodation booking
  - Bank transaction

[1]Organization for the Advancement of Structured Information Standards

Each of them is supposed to provide for a specific step of the vacation planning procedure.

- After further analysis, they figure out that the trip planning service may be split into other sub-components:
  - Flight booking
  - Car rental
  - Bus ticket purchasing

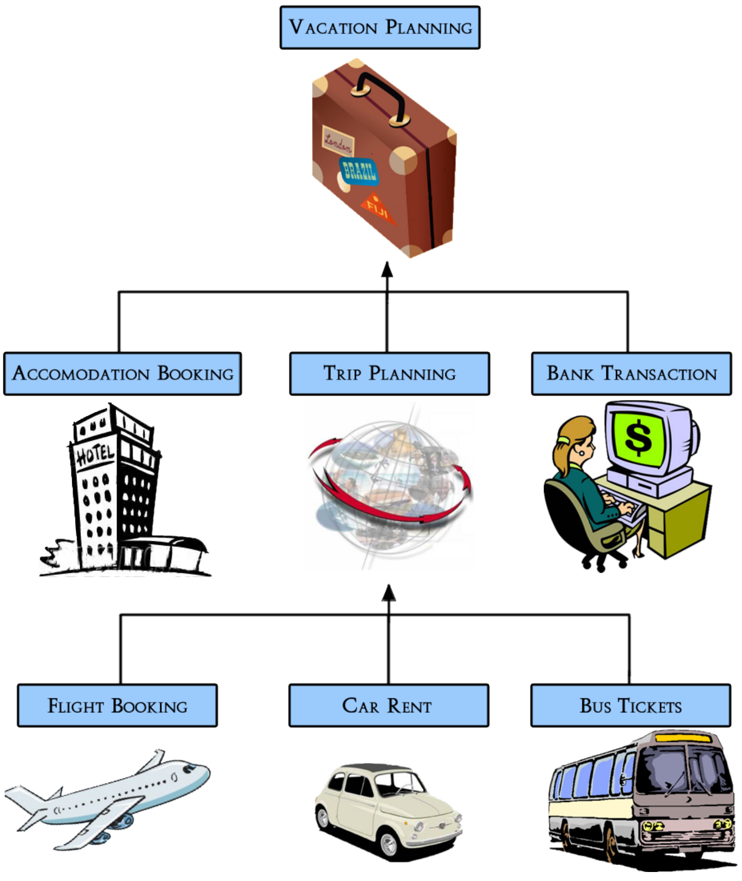


Figure 1.1: Concern Separation - Second step

Each of the components the company identified can work as a stand-alone unity. They offer various interfaces to interact with and the collaboration between them can lead to the expected solution. They are the so-called *services*:

- each one provides for a specific set of functionally related capabilities
- they can work as an independent part of the system
- they are agnostic of each other and of the greater problem they are going to solve

But, when composed together, they can help achieving a more complex goal.

### 1.3.2 The Service Oriented paradigm

There have been many programming design paradigms to date, each of them conceived to achieve a specific goal. For instance:

**The object oriented paradigm** , conceived to bring modularity to the code and make it reusable and easily maintainable.

**The aspect oriented paradigm** , stating that the system logic should be broken into distinct cross-cutting concerns.

The aforementioned paradigms are considered some of the roots[27] of the design paradigm that started to gain a lot of attention in 2005 [42]: *the service orientation.*
The Organization for the Advancement of Structured Information Standards provides a definition of this concept:

> *A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.*[64]

This paradigm is based on the principle that a large solution should be partitioned into smaller capabilities, each designed to solve an individual problem.

In the context of the paradigm, the solution capabilities (or units) are referred as *services.*
In order to fulfill the *service orientation* idea, such services have to hold a set of design principles[27]:

**Standardized Contract:** each service needs to expose a contract to describe its purpose and capabilities.

**Loose coupling:** service consumer, service contract and service implementation must not be strongly dependent to each other. These three entities have to be weakly tied together. A strong dependency between service contract and service implementation would imply a big effort to evolve, maintain or change certain aspects of the logic behind the service interface. As well the service consumer has to be independent from the service provider: the service client should be uninfluenced and unaware of the internal features of the service provider (such as operative system or programming environment).

**Abstraction:** the service logic has to be hidden behind the interface contract. This allows to preserve the previously mentioned *loose coupling* between service interface and service implementation.

**Reusability:** when implementing a service, it is necessary to keep it agnostic of its functional context. This means that it has to be considered as a long lasting resource, ready to be used whenever its capabilities are required.

**Autonomy:** in order to have a reliable service (necessary for supporting a real world production environment), the service logic must have significant control over its environment and resources.

**Statelessness:** considering that a service may be reused lots of times by many different consumers, it may be undertaken to a great scalability demand. Statefulness requires an adequate surrounding technology, so, *when possible*, a service should be kept stateless.

**Discoverabiliy:** in order to improve reusability, it is mandatory that a service can be in a first place discovered. This goal can be achieved by consistently describing the service. To do that, the service contract should be extended with functional meta-data as well as meta-informations related to the QoS[27]. This way, the meta-informations can be retrieved (after being published somewhere) in order to be analyzed either manually or automatically. If the service results fitting with the consumer need, it may be reused and integrated.

**Composability:** service oriented computing relies on the possibility to create a solution by separating it into small problems and by identifying the components solving them. In order to achieve this result, components (services) have to be capable of joining a composition, even if they are not immediately enlisted in it.

These design principles are the archetypes of a system based on a Service Oriented Architecture.

### 1.3.3 The Service Oriented Architecture

In the previous section the main principles defining a *Service Oriented Architecture* have been described.
This sort of architecture is based on the interaction of three primary parties[8]:

- **The service provider**: the entity providing for the service. It implements different services and exposes certain interfaces to allow their use.
- **The service consumer**: the client who utilizes the service by invoking its interfaces.
- **The service broker**: the entity offering the discovery capability. It can be a public registry, or an agent acting on behalf of the service consumer.

In SOA, all the resources can be linked on demand. They are available to consumers allowed to use them, be them within an enterprise or across multiple enterprises. There are different business-aligned IT services that collectively fulfill an organization's business processes and goals. In order to achieve the more complex goals, it is necessary to orchestrate these services in a service choreography.

## 1.4 SOA Technologies

### 1.4.1 Early Technologies

There are many ways to implement a service based system to date. Some of the ideas and technologies currently available were born even before the idea of SOA was conceived.
One of the first models that might have fit a service oriented architecture was

the Remote Procedure Call, whose idea goes back at least up to 1976. The definition given by SGI[2] is:

> *Remote procedure calls are a high-level communication paradigm that allows programmers to write network applications using procedure calls that hide the details of the underlying network. RPC implements a client/server system without requiring that callers be aware of the underlying network.* [41]

This enables an application to perform a procedure call to a remote machine unaware of its underlying architecture and platform. And this is of course a first step to concretise the interoperability feature required by the service orientation paradigm. Even the client/server structure maps the consumer/provider parties required by SOA. Based on this model, lots of technologies subsequently born:

- Distributed COM (DCOM), a proprietary technology by Microsoft. It allows components distributed across a network to communicate, offering the communication substrate under Microsoft's COM+ application server infrastructure.
- Remote Method Invocation (RMI), a Java programming interface that allows communication between objects (in an object-oriented context), in a java programming environment.

Both of these technologies are weakly following two of the main principles of the service orientation paradigm:

- *strong coupling*: service consumer depends on the service provider. DCOM imposes the underlying existence of a Microsoft platform. Java RMI requires objects to be written in Java.
- *weak composability*: this is connected somehow to the coupling issue. In case the service is asked to join a choreography, it comes difficult to integrate it to the current composed system if platform/programming language requirements exist.

There are indeed other technologies concretising some of the concepts of the service oriented paradigm (e.g. CORBA, DDS), but it would be out of the scope of this document to carefully analyze them.
Pan Li et al. provide the following consideration concerning SOA:

[2]Silicon Graphics International, `http://www.sgi.com/`

*Service oriented computing (SOC) is a new and promising paradigm for open, distributed applications. Web Services, along with a group of standards, has become de facto integration technology for realizing SOC.*[60]

Looking at the literature it is in fact quite common to find the concepts like SOA, SOC etc. pulled-over terms like Web Services, SOAP, WSDL and so on. In the following section Web Services will be then briefly explained so to provide a glimpse on how the Service Oriented Vision is turning into a concrete form.

### 1.4.2 Web Services

The term *Web Service* has gained a bit of semantic confusion over the years. There are different definitions about what a Web Service is supposed to be. The W3C definition of Web Service is:

*There are many things that might be called "Web Services" in the world at large. However, for the purpose of this Working Group and this architecture, and without prejudice toward other definitions, we will use the following definition:*

*A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*[33]

The W3C agreed definition for Web Service involves the use of technologies such like SOAP and WSDL. Probing the Internet, it is nonetheless possible to discover other quite different meanings of the term. Let us for instance show this piece of article by Alex Rodriguez, an IBM software engineer:

*Representational State Transfer (REST) has gained widespread acceptance across the Web as a simpler alternative to SOAP- and Web Services Description Language (WSDL)-based Web Services. Key evidence of this shift in interface design is the adoption of REST by mainstream Web 2.0 service providers - including Yahoo, Google,* *and Facebook - who have deprecated or passed on SOAP and WSDL-based interfaces in favor of an easier-to-use, resource-oriented model to expose their services.*[78]

REST-ful Web Services seems then to be a valid alternative to the SOAP based ones.
This "role clash" issue arises probably because REST based WS have gained attention not a long time ago and so they are still lacking of standard definitions to be described at a W3C level. Nowadays REST-ful Web Service users (developers) are increasing, but SOAP remains the de facto standard for enterprises. They share the main critical feature of the data transport: they both usually use HTTP as a transportation protocol, describing data through an XML document (even though also JSON is possible with REST-ful Web Services).
The following sections will give some further details about these two technologies.

#### 1.4.2.1 SOAP based

The SOAP based Web Service architecture is usually composed by the three main entities a SOA requires: service provider, service consumer, service broker. The entities illustrated in diagram 1.2 are:

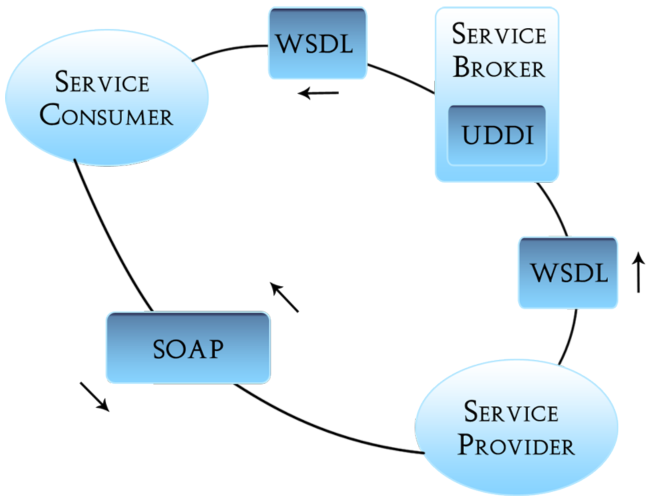


Figure 1.2: SOA High Level Architecture

**SOAP:** provides a standard, extensible, composable framework for packaging and exchanging XML messages.
A SOAP message represents the information needed to invoke a service or reflect the results of a service invocation, and contains the information specified in the service interface definition.[30]

**WSDL** WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate.

**Service Provider:** It is the actual owner of the service. It exposes the web interfaces needed to invoke the service functionalities. It communicates with the service consumer by means of SOAP messages. In a nutshell, it is composed by three parts[85]:

- a listener to receive the message,
- a proxy to take that message and translate it into an action to be carried out,
- the application code to implement that action.

It is possible, then, to implement the service logic with any programming language for which a SOAP implementation library exists.

**Service Consumer:** It represents the other communicating peer. As for the *service provider*, the *service consumer* needs a proxy in charge of converting procedure calls in SOAP messages to send over the net to the *service provider*. Before that it clearly has to *discover* the needed service and, after having agreed on the contract, it has to refer to the WSDL description in order to know what web interfaces are publicly disclosed.

**Service Broker and UDDI:** Here each service provider publishes its own service descriptions (WSDL) and each service consumer can discover Web Services by accessing the service broker registry. UDDI is an acronym for Universal Description, Discovery, and Integration. This is an XML-based registry whose specification is provided by the OASIS Group[3]

[3]`http://www.oasis-open.org/committees/uddi-spec/`

#### 1.4.2.2 REST based

RESTful Web Services are based on the Representational State Transfer architectural style. It is a client/server based architecture, whose main principle states that requests and responses has to be based on the transfer of representations of resources. In a Web Service context, the resource is represented by the Web Service itself and it is identified by its URL. The client can access the resource by invoking the corresponding URL. The principles of RESTful Web Services are[78]:

- Use HTTP methods explicitly.
- Be stateless.
- Expose directory structure-like URIs.
- Transfer XML, JavaScript Object Notation (JSON), or both.

As stated in the first point, the operation the service requester can perform are represented by the HTTP methods, such as[92]:

- **GET**, to perform a read operation
- **POST**, to perform a write operation
- **DELETE**, to perform a deletion
- **PUT**, to perform an update

The “REST way” of invoking services can be better explained by means of an example: let us consider the situation where a service consumer needs to add a user to a registry. The usual way of doing this through an HTTP request is:

```
        GET /adduser?name=Ford+Prefect HTTP/1.1
```

The REST way would be instead:

```
POST /users HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
        <user>
                <name>Ford Prefect</name>
        </user>
```

As it is possible to see, a REST-ful service uses HTTP verbs already defined in the protocol, without reinventing an additional layer.
Regarding the WS contract, there are no spread methodologies to date. Usually the URL is self-explanatory for a human observer. For instance, a typical way of describing a service is[78]:

```
http://www.myservice.org/discussion/{year}/{day}/{month}/{topic}
```

This way the service interface results quite intuitive, even without strict rules to code it. But it could not probably be processed by an automated agent, like WSDL makes possible to do. Even though a WSDL-like description language has been defined for REST-ful Web Services (WADL[4]), it is rarely adopted. The main cause is probably a cultural reason: REST-ful Web Service are supposed to be light and easy to use and so a WSDL-like document would not be the simplest way to describe them.
Another key-difference between RESTful Web Services and SOAP ones is the absence of popular REST Web Service registries.

## 1.5 Use cases

Use-cases for SOA usage and Web Services can be found in [29][31][32][68], but many others are also available all over the Web.
We are not going to list all the possible scenarios involving a Service Oriented Architecture, because it would be out of the topic of this work.
Analyzing the literature related to Web Services and SOA, and referring to [31], there are however 2 main categories almost all the scenarios belong to:

**Static use** :
This category involves all those use cases where the developer, at programming time, discovers the service providers and selects the required Web Service prior to the invocation. Then the developer uses the meta-data and description to create the Web Service invocation.

**Dynamic use** :
In this case, the Web Service invocation is performed after the system is deployed. This means that the discovery, the selection and the ensuing invocation of the Web Service are performed by a software agent at runtime.

[4]`http://www.w3.org/Submission/wadl/`

A further separation can be applied depending on the dominium: enterprise or non enterprise.
Enterprise-level use-cases (those that actually created the need of a Service Oriented Architecture), involve all the scenarios where a company A needs to conduct some sort of electronic business with company B. For instance, A might be willing to automate the exchange of business documents with B. Thus, to loose the integration issues and to improve automation, they may establish a WS-based infrastructure for the communication.
Web Services can be adopted, then, outside the enterprise, targeting more generic consumers. For instance, to compose the underneath architecture of a distributed system whose results can be potentially requested by any user (unaware of the underlying architecture). A common example for this case is the Virtual Travel Agency provided by W3C[29].

## 1.6 Trust in SOA

There have been many issues discussed about Web Services over the years, regarding their implementation, their founding principles and so on. But there is still a concern that would need to be thoroughly investigated: how can a service be trusted? What is the right choice when it comes to decide the best service among plenty of similar ones?
Let us provide an example by means of the well known Virtual Travel Agency (VTA) scenario:

- Alice is a software developer for Encom Enterprise;
- she's asked to develop a VTA system, a service helping the end-users through all the necessary steps to plan a trip;
- she decides to break down the system into its smaller capabilities:
  - flight booking
  - accommodation booking
  - bus ticket purchase or car rental
  - payment
- since there are plenty of Web Services providing for the identified features, she (or an agent on her behalf) has to choose the right one;
- **Issue**: which service is the **right** one?

- Alice did not use any of the available services in her career, so she picks them up just relying on few descriptions and her common sense (or the agent does that, without any common sense);
- after one month, the company providing the *flight booking* service has to temporary shut down the service: the Web Service Alice picked up turned out to be not scalable for the intense traffic the VTA was having.

As the example highlights, selecting the right service does not include only the problem of discovering services on the basis of what a service can do (functional properties), but also on how well a service can do (nonfunctional properties), evaluated according to some non-functional metrics.
Another example could involve Alice in the role of the final consumer instead of the developer. She may have her personal *VTA software agent* on her smartphone that, based on her needs, searches the best offers for a trip. Alice is then provided with a list of possible Web Services where to book her trip, but the problem of which one she should *trust* still remains.
The issue we are going to analyze throughout the document concerns the problem of *automated trust provisioning* in SOA environment.
The VTA scenario will be used as a running example to discuss the *trust provisioning issue* in SOA henceforth in this document.

Chapter 2

# Trust in SOA - State of the Art

In Chapter 1, we analyzed the background of Service Oriented Computing, an emerging paradigm for distributed computing, where WSs represent the bricks of a Service Oriented Architecture.
We discussed about how nowadays this vision is being concretised, highlighting definitions and technologies involved. At the end we highlighted that there is still a concern that would need to be thoroughly investigated: how can a service be trusted? What is the right choice when it comes to decide the best service among plenty of similar ones? In this chapter we are going to show how in literature they tried to answer this question, providing a survey about the state of the art of the Web Services (WS) trust provisioning. In the conclusive part of the chapter, the sum-up of the considerations is provided, focusing on the limitations of the current approaches and laying the foundations for a new approach.

## 2.1 Terms confusion

One of the major point of confusion concerning the current discussions about trust/trustworthiness in SOA is related to the meaning of terms. In section 2.3, many different literature studies are analyzed, but in most of them the word *trust* and the word *trustworthiness* are used with the same acceptation or with different meanings in different works. The two terms have a precise meaning and *trust* should not be confused with *trustworthiness*. This will be thoroughly discussed in Chapter 3. In the meanwhile the two terms will be treated as already did so far in literature.

## 2.2 Sources of trustworthiness

In the studies present in literature, trustworthiness of WSs can be drawn by many different sources, as we will better see in 2.3. A considerable part of them depends on the QoS features of a WS. For this reason, we provide in this section a brief explanation of what this metrics are and how they contribute to the non functional characteristics of a service.
All the adopted metrics boil down to the QoS metrics for WSs identified by W3C (figure 2.1). As shown in the diagram, there are three main groups of QoS

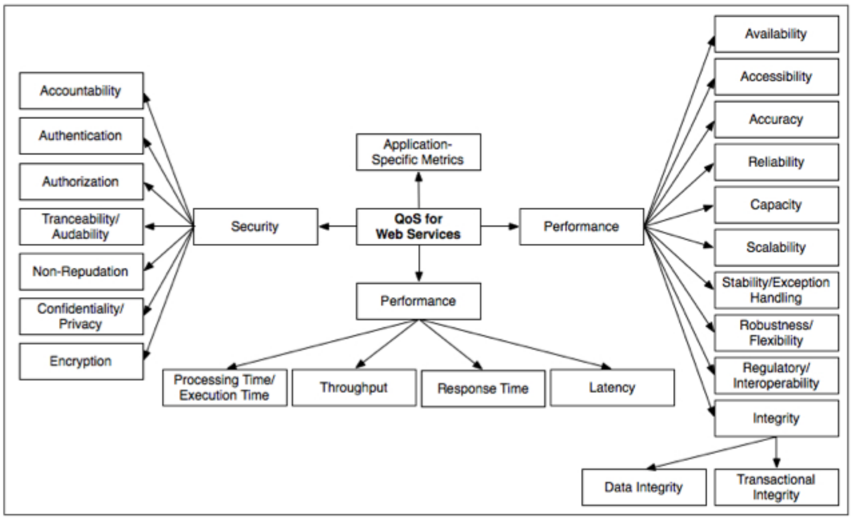


Figure 2.1: QoS metrics according to the W3C Consortium

features:

- *Security*: the security features can be statically derived, because, when they exist, they are part of the WS contract.
- *Performance*: these features depend on the overall past experience with the service usage. They can be derived by a consumer constantly analyzing and evaluating its experience with the given WS.
- *Application specific*: The aforementioned metrics are common ones, which can be applied to most domains. Since WSs are so diverse, it is impossible to capture all QoS metrics for all of the domains in a single model. Therefore, the fourth group, "application-specific metrics", is reserved for the metrics that are specific for a certain domain[93]. For instance, a Flight Booking system may be evaluated depending on the amount of times it manages to find the cheapest flight among the main flight companies at a given moment. Or another flight booking system may receive a higher grade if it allows users to choose multiple flights to compose a single journey. These features, then, may require or not the consumer past experience to be evaluated.

The *Security* features are what Rasmussen & Jansson define as "hard security mechanisms"[76]. This means that they are not evaluated in a social fashion, but they depend on the system implementation.
*Performance* and *application specific* metrics can be drawn only after some interactions with the service, and they fit in a "soft security" system. Some studies include the *reputation* as part of the QoS features of WSs, as suggested for instance in [104] by Bo Zhou et al. But there are diverging opinions about that: other studies like [98] consider the reputation an QoS separated concepts. We will provide a better explanation of the Quality of Service definition issue throughout Chapter 3.

## 2.3 Suggested approaches

There has been a growing literature on studies aimed to address the problem mentioned in Section 1.6. Many different frameworks and protocols have been suggested in order to provide the end user with some sort of trustworthiness clues. In this chapter we aim at grouping and classifying the whole "jungle" of studies, according to different points of view. Each category identifies issues and advantages shared by all the belonging works.
This preliminary analysis will highlight the main points of discussion regarding the topic of trust provisioning in SOA, helping the ensuing definition of a new approach to be adopted in our framework. Part of the concepts treated in this section of our work derives from previous studies such like [9][23][16].

The categorization can be summarized in figure 2.2:

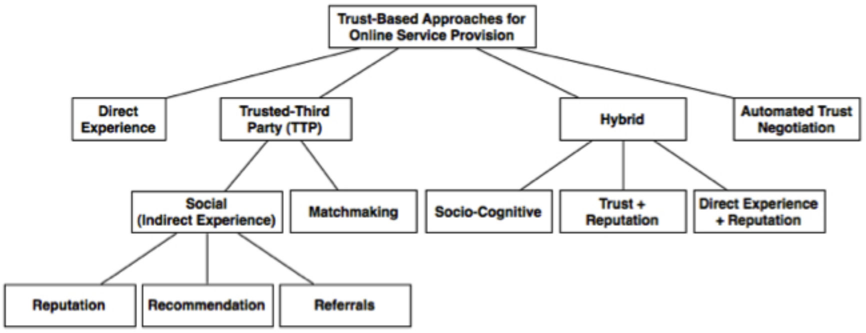

Trust-Based Approaches for Online Service Provision
Direct Experience
Trusted-Third Party (TTP)
Hybrid
Automated Trust Negotiation
Social (Indirect Experience)
Matchmaking
Socio-Cognitive
Trust + Reputation
Direct Experience + Reputation
Reputation
Recommendation
Referrals

Figure 2.2: Classification of Trust Approaches for Online Service Provision[23].

Even though the main effort of this state of the art investigation has been to devise a rationale-base categorization (as shown in Fig. 2.2 and thoroughly discussed later on in this Chapter), it is necessary to provide a more general classification based on architecture of the different frameworks. In fact, even the architectural choices can result in a more or less adequate solution for this context.
The systems studied in literature are mainly based on 2 architectures: centralized or distributed. Thus, Even if each single approach applies different rules for the entity-to-entity communication or uses different algorithms to compute the trust, the architecture-dependent behaviour can be generalized as follow:

**Definition 2.1 (*Distributed trust provisioning approaches*)**
The trust scores of WSs / service providers are computed/derived after having directly communicated with other peers in the system.

**Limitations:** in the context of SOA it is not really possible to generalize the shortcomings deriving from a distributed architecture, because they are tightly connected to the specific approach. Anyway, in general, a common drawback is the system setup and start-up effort.

**Definition 2.2 (*Centralized trust provisioning approaches*)**
The trust scores of WSs / service providers are provided by a central authority with the responsibility of computing/collecting them.

**Limitations:** central authorities are a single point of failure and thus can exist only under rigidly constructed and administered computational environments, in particular considering the capacity demand of a SOA environment. Another technical limitation resides in the possible alteration of the ratings (collusion or retaliation). Moreover, a centralized trust authority *can never be a good enough recommender for everyone: different entities should be allowed to make up their own mind*[5].

Further advantages and drawbacks will be discussed in the following sections according to each approach.
These initially outlined categories embody all the approaches that we are going to describe in the following sections. Each of them is designed around either a distributed architecture o a centralized one. Some approaches envelope works conceived on a both distributed and centralized architecture.

### 2.3.1 Direct Experience

This class of approaches is based on *presumptions drawn from the service consumer's own direct experience with the target service*[7]. The rationale is that the trust can be build upon some quality parameters that depend on the service behaviour in the course of time. This means that the service can be trusted if the consumer past experience with that service results complying to the given trust policies. A model following this principle is the one suggested in *A Cognitive Trust-Based Approach for WS Discovery and Selection*, where experience is defined as follow:

> *An Experience is the knowledge gained after having a transaction with a service. The experience with a particular service is stored in a private repository as a set of values termed as Quality of Experience (QoE). The term (QoE) is defined as how the user feels about how a service was delivered, relative to his expectations and requirements.*[7]

Jonker and Jan Treur present an analysis of models for the dynamics of trust based on experiences[43]. They try to investigate the founding principles governing the evolution of an agent's trust towards a service. In order to provide some evolution or update function, they firstly had to define an initial trust, that is currently the problem of the trust-by-experience approaches. They then assume either unconditional trust or unconditional distrust where no direct experience is available. In [103] the authors describe a layered framework conceived to manage trustworthiness at seven levels. However, as the authors themselves point out, *the quality of the model to be built is fully dependent on the experience of the practitioners*, that, as described below, is the main weakness of the trust-by-experience approaches.

**Definition 2.3 (Trust by Direct Experience)**
A service consumer trusts a service because of his good past experience with the service.

**Definition 2.4 (Initial trust)**
The assumed trust belief of a consumer when no direct experience is available.

**Limitations:** This approach does not suit large open systems where anyone can publish its (malicious) code, since it does not allow to trust a service before its execution. Moreover, whenever an *unconditional distrust* approach is used, brand new services may be not considered trustworthy even if conforming to the needs.

**Scenario 1:** Alice has no way to select the flight booking service according to her trust preferences. She might decide to blindly select the WS, taking the risk of invoking the it although there is no evidence of its trustworthiness. This forces Alice to accept not only the WS inherent risk of prior performance (i.e., to pay for services and goods before receiving them) but also the risk of blind (i.e., untrusted) execution, since she is going to invoke the service without trusting it.
On the other side, she might decide to not trust a service just because of a slightly unwelcome policy (e.g. the service demand the consumer's address to be accessed), even if the service was actually compliant with all the needs of Alice (making eventually the policy more acceptable).

### 2.3.2 Trusted Third-Party Approaches

Truste Third-Party (TTP) approaches are based on the idea that the service consumer can rely on a third-party in order to obtain a trust value of a given service. *Third-party* may refer to different sort of entities:

- a trusted central authority
- members of a community, that can be represented by other service consumers or interacting agents (further details later on)

The underlying assumption in these approaches is that consumers must trust the third party they decide to consult. This category is further split in two sub-categories: social and matchmaking approaches. In both of them the final decision is based on the assessments provided by the TTP. The difference lies on how the assessments are computed.

#### 2.3.2.1 Social (Indirect Experience)

The trust evaluation towards a WS is forged by a cooperating community whose members have directly or indirectly interacted with such WS. In order to be effective, each community member has to continuously review the services (and the service providers) he is using. The global evaluation is not necessarily calculated by the community members themselves, but might be the result of a centralized information retrieval applied on member-supplied information.
In literature we can find three different *social* approaches:

- **reputation**, based either on a centralized authority or on a distributed P2P system
- **recommendation**, based on a central authority
- **referral**, based on community of interacting agents

**Reputation**

The definition of the term *reputation* from The English Oxford Dictionary is:

> *a widespread belief that someone or something has a particular characteristic*

In a SOA context, the rationale is that a service is trustworthy as long as the community has a good opinion about it. The reputation system is responsible to collect ratings about users, services and service providers from members in the community.
The global opinion can be modeled either around the QoS parameters described in Section 2.2 or depending to what degree WSs adhere to the contract. However, the parameters used to rate a service, user or service provider are not influencing the rationale of a TTP reputation system: an individual's subjective trust on a service is derived from the reputation of that service or, in other words, from the direct experience of someone else. This leads to the following definition:

**Definition 2.5 (Trust by Reputation)**
A service consumer trusts a service because of its good reputation.

**Limitations:** another shortcoming is that trust relies on past information from other members of the community. A natural problem arises in case of new services. For example, when a service initially registers for business, no other consumer has interacted with it and consequently no information exists about its past behaviour and questions about its trustworthiness are left unanswered. This can be defined as the *new WS ramp-up issue*.

**Definition 2.6 (New WS ramp-up issue)**
A new WS takes time before being adequately evaluated.

**Scenario 2:** Alice found a service for booking flights with a good contract under the functional point of view. She does not have any experience though with that WS, so she looks for some reputation score of the service. Unfortunately, the service is new and no reputation scores are available. Alice is still in a vulnerable position, under the risks of untrusted execution.

**Limitations:** Another shortcoming is that the effectiveness of any reputation system relies on the number of members in a community and on their behaviour. In particular, the fewer the members in a reputation system, the more inadequate the ratings provided by the systems. This issue gives life to the *community-bootstrap* problem.

**Definition 2.7 (Community bootstrap issue)**
A community-dependent system is unlikely to provide good quality results as long as the community is small or not really active.

**Scenario 3:** Alice tries to query a reputation provider for getting a trustworthiness value of the WS she found. The value is low, so she decides to do not trust the service. Nevertheless, that service was evaluated by a very small number of consumers, non statistically relevant for a reliable rating.

A major distinction between different reputation systems is outlined by the base architecture: centralized or distributed [93][46].

**Centralized**
Most of the studies suggests centralized approaches[65], typical of e-commerce

web sites[1]. In those systems a central authority is responsible to collect all the ratings from other members in the community (e.g. QoS data from WS consumers, in our case) who have had direct experience with a specific service or provider. The authority uses these ratings to derive a reputation score for the service and makes it publicly available to future, potential consumers. The central authority is then responsible for

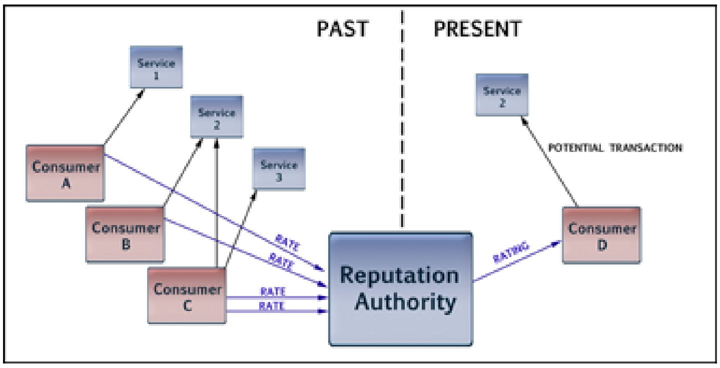


Figure 2.3: Centralized reputation system

1. authenticating the users,
2. recording, aggregating and revealing ratings,
3. owning ratings.

In [65], a central QoS registry is deployed to collect and store QoS data from WS consumers.

**Definition 2.8 (Centralized reputation system)**
The reputation value of a WS / service provider is owned by a central authority.

**Limitations:** A technical limitation, as it may happen in such a kind of reputation system, resides in the possible alteration of the ratings (collusion or retaliation). Moreover, a centralized reputation system suffers of all the shortcomings listed in Section 2.2 for approaches based on a centralized architecture.

**Distributed:** two notable systems exploiting a distributed reputation community are EigenTrust[48] and the PeerTrust[96]. These are based on a

[1]e.g. `http://www.ebay.com`

system without a central authority. Each member of the community (be it an agent or a human) records his own opinion about a service to make it available to the other members. If one of them needs a trust evidence of a given WS, he tries to request ratings to all the possible members he can reach (if they had some previous experience). The reputation grade is then calculated as a function of the obtained ratings.

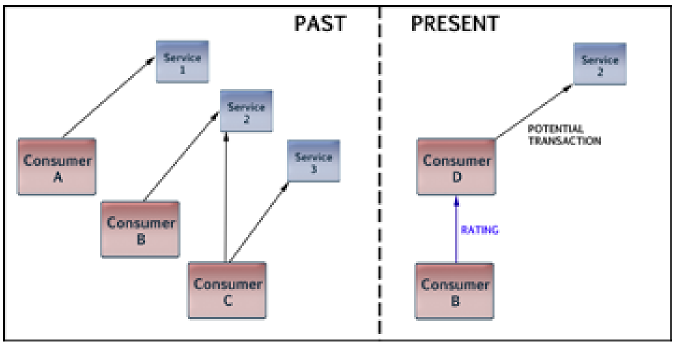


Figure 2.4: Distributed reputation system

**Definition 2.9 (Distributed reputation system)**
The reputation value of a WS or of a WS provider is computed after having received the reputation scores from other members of the community.

**Limitations:** Issues explained in Def. 2.7 and 2.6 are still present in this solution.

**Scenario 4:** Alice needs to obtain the best flight booking service among those who are publicly available. Her software agent tries to query all the neighbour agents for the reputation grade of a service in the P2P net it belongs to. The agents (based on theirs and their neighbours past experience) return the answer and, after having calculated the reputation score, Alice's agent decides to discard the service. Alice still does not know whether to trust or not the agent because the WS is a really new one and the P2P community might still be missing some real long term experience with that service.

Many studies have addressed the new WS ramp-up issue defined in Def. 2.6. The question is: how the community should repute a new service (or new provider, depending on the evaluated subject)? There have been many suggestions regarding the choice of the right starting reputation score. In the Sporas system suggested by Giorgios Zacharia et al.,*new users start with a minimum reputation value, and they build up reputation during their activity on the system. The reputation value of a user never falls below the reputation of a new user*[102]. The authors of the Dirichlet algorithm[46] (conceived for P2P sharing network) state that *it is possible to track the average reputation score of the whole community, and this can be used to set the base rate for new agents, either directly or with a certain additional bias*[46].

**Limitations:** In general, when the starting reputation is low, the new WS (or provider) is underestimated and, even with a good actual behaviour, it may be discarded a priori. Whenever a new WS receives an initial reputation score higher than the minimum, this can be exploited by malicious users by continuously subscribing and unsubscribing to the system in order to keep having a "non zero" reputation value.

**Scenario 5:** Alice's software agent starts discovering new flight booking WSs on behalf of Alice. It queries a TTP (be it either a P2P community or a centralized authority) who states that the chosen service has a non zero grade. Alice still does not know whether the service is trustworthy or not, because the service might belong to a malicious provider just subscribed to the community.

### Recommendation

Recommendation systems [9][5] aim at making a prediction of a consumer's needs or interests. In its common formulation [6], the recommendation problem is reduced to the problem of estimating ratings for the items (such as services) that have not been seen by a consumer. Intuitively, this estimation is usually based on the ratings given by this consumer to other items or on the ratings that similar users provided for the targeted items. Once it is possible to estimate ratings for the yet unrated items, then the system can recommend to the user the items with the highest estimated ratings.

**Definition 2.10 (Trust by Recommendation)**
A service consumer trusts a service because of some recommendations got from a trusted authority.

In general, *recommendation-based systems* work as good as wide and rich the knowledge of the system is. In other words, it is necessary to know both the community and the user requesting the service in order to produce reasonable evaluations.

**Limitations:** The key weakness of this family of approaches, as for reputation systems, still lies in the rationale of the approach: they rely on the existence and good working of a community that provides ratings to the centralized recommender system. In large open service-oriented systems these assumptions are too strong, leaving a consumer to a vulnerable position in case he does not belong to any community or the community is so poor that does not provide a significant ratings. Finally, recommender systems are conceptually centralized and the weaknesses discussed for centralized systems are still valid (Section 2.2).

**Scenario 6:** Back to our scenario, Alice's agent does not belong to any recommender community. Therefore, Alice is still left in a vulnerable position, under both the risks of prior performance and untrusted execution.

Recommender systems are usually classified into five categories, according to how recommendations are computed [9][23][16]:

**Content-Based Filtering** : the consumer is recommended items similar to the ones he preferred in the past; it is a static approach for selecting items by filtering web sites or documents in terms of the words that occur in them. For instance, this approach could be applied to services by indexing their textual descriptions. But this approach is very primitive and would be a step backward from current WSs standards (which involve formal structured description of their functionalities);

**Collaborative Filtering** : the consumer is recommended items that people with similar tastes and preferences liked in the past; Collaborative Filtering (CF) represents the most widely used recommender method, for instance in e-commerce sites such as Amazon[2]. In CF, user's ratings for different items are stored centrally and these ratings are often simply captured as the products a given user purchased [38]. If two users rate a set of items similarly, they share similar tastes and for this reason they are neighbours in the jargon. This information can then be used to recommend items that one participant likes to his or her neighbors. In other words, a user is given recommendations based on the ratings by other users who are similar to the given user, that is who have similar subjective tastes.
A simple example of how a recommender system works in general is shown in Fig. 2.5. If Alice and Bob both bought books A, B and C and Alice

[2]`http://www.amazon.com`

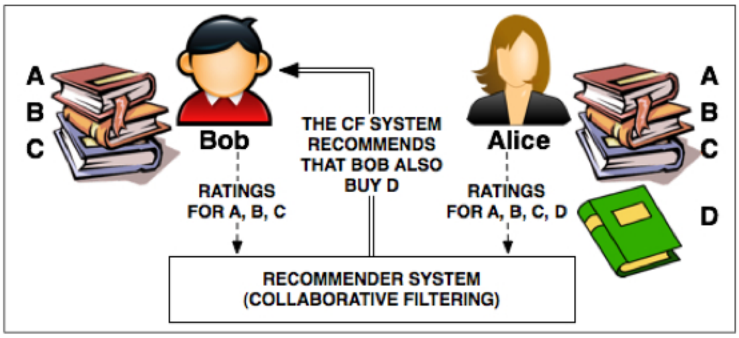

A
B
C
Bob
THE CF SYSTEM RECOMMENDS THAT BOB ALSO BUY D
Alice
A
B
C
RATINGS FOR A, B, C
RATINGS FOR A, B, C, D
D
RECOMMENDER SYSTEM (COLLABORATIVE FILTERING)

Figure 2.5: Basic idea of Collaborative Filtering[24].

bought also book D, then a CF system may recommend Bob to buy D. The implicit assumption underlying CF systems is that different people have different tastes and rate things differently according to subjective taste. Note that this represents the key difference with respect to reputation systems [47]. Indeed, reputation systems are based on the seemingly opposite assumption: all members in a community should judge the quality of a product or service consistently. In other words, CF takes ratings subject to taste as input, whereas reputation systems take ratings assumed insensitive to taste as input.

**Limitations:** While recommender systems seem working well in e-commerce sites to buy products, there are some limitations of applying this approach to service selection [38]. First of all, the assumption that someone purchased a service does not mean that they liked it. Then, in the SOC vision, services are distributed and advertised by a registry or broker. This entity does not provide the service that it is recommending and may have little to say about its trustworthiness. For instance, a registry would not have any control on the actual service interaction, whereas an e-commerce site would know that a product was shipped correctly.

**Demographic:** [77][54][71] This sort of recommendation systems aims at categorizing users analyzing their personal attributes and at making recommendations based on demographic classes. This can be achieved by starting classifying some given examples, like consumers and WSs rating; after that, whenever a potential WS consumer submits a request to the system, his demographic information are compared with the already given one in order to obtain a potentially preferred WS (by means of a rank).

**Limitations:** In order to make the system capable to provide the match to the requesting user, it is necessary both to have a starting knowledge to build a demographic classifier and to gather the user demographic profile. This can be done by asking the new user to disclose some sort of personal information or by mining his profile out of his interaction with the system. This is clearly not a proper idea if applied to a SOA-based open system, where the user may probably not want to disclose his information and (in the other case) may need to get recommendation since the beginning of his interaction with the system, without waiting a long time before having his demographic profile computed.

**Scenario 7:** Alice decides to join a community based on a demographic recommendation system. In order to join, she is asked for some sensitive information of the company is going to use the service. Such information are the company name, working field and currently used WSs. The company is not really willing to disclose these details. So Alice does not provide any information, but she still cannot obtain any suggestion from the system concerning a *flight booking* WS because there is no demographic knowledge of the consumer (Alice on behalf of her company).

**Utility-based:** [34] The rationale of a utility-based recommender is that each object of a set (WSs in our context) has a defined utility value for each user. The utility-based recommenders derive the utility function of the user in order to compute his profile. Then the profile is used to find the most fitting objects[34]. The benefit of utility-based recommendation is that it can factor non-product attributes (such as WS QoS parameters) into the utility computation.

**Limitations:** As for the demographic recommender, the user needs to input his own utility function, facing the same problem as the demographic and collaborative filtering recommenders. In a centralized recommendation system it might be a too strong assumption to think the user is willing to unconditionally share his preferences with third parties.

**Scenario 8:** Refer to the Scenario 7.

**Knowledge-based:** [90][83][81] Knowledge-based recommenders attempt to suggest objects based on inferences about a user's needs and preferences. In some sense, all recommendation techniques could be described as doing some kind of inference. Knowledge-based approaches are distinguished in that they have functional knowledge: they have knowledge about how a particular item meets a particular user need, and can therefore reason about the relationship between a need and a possible recommendation.

**Limitations:** Knowledge-based recommenders have the intrinsic problem of all knowledge-based systems: they need a starting knowledge, both of the community and of the users. This is, for the same reason mentioned above, a problem in a system where users need to find the right component for their service oriented system even without necessarily having a past interaction with the community.

**Hybrid Approaches** : In a survey by Robin Burked[16], the hybrid recommenders are introduced with this definition:

> *Hybrid recommender systems combine two or more recommendation techniques to gain better performance with fewer of the drawbacks of any individual one. Most commonly, collaborative filtering is combined with some other technique in an attempt to avoid the ramp-up problem.*

In a nutshell, hybrid approaches just consist of the joint collaboration of some of the approaches mentioned above, either by combining the final results of each of them or by making them interact during the mining process.

**Limitations:** Each problem should be treated with a different combination depending on the domain, the available data and so on. Considering that the hybrid recommendation approaches are still a matter of research, it is difficult to identify one system capable of achieving the required goal to date. It should be necessary to conceive a specific hybrid recommender for a SOA trust provisioning system and than prove its reliability.

Those typologies of recommendation system share one main weakness: the system needs many information about the users in order to provide useful evaluation. This can be achieved by either asking the users to disclose maybe sensitive information (as we have seen this does not suit the SOA environment) or by mining them out of the interaction of the users with the system, that would require a long time. This leads to the following issue definition:

**Definition 2.11 (New User ramp-up issue)**
A new user needs to interact with the trust provisioning system in order to receive good quality results.

Moreover, the well known issues of the social systems are still present: community bootstrap and new service ramp-up (mitigated in the hybrid recommender). Finally, the recommendation systems are conceptually centralized (Section 2.2).

## Referral

A common weakness of most recommendation and reputation mechanisms lies in their being conceptually and architecturally centralized: a single authority is responsible to collect, aggregate and present all the ratings. To address this limitation, referrals [84][100] have been proposed as a decentralized approach based on online communities and software agents technologies. An online community is a set of interacting members (or principals in the jargon) representing people, businesses or other organizations. The members of a community provide services as well as referrals for services to each other. Referrals may be provided proactively or in response to requests. Members are assisted by software agents to help them manage their interactions. Software agents are persistent entities that can perceive, reason, act, and communicate [84]. Agents represent different members and assist them in evaluating services and referrals provided by others, maintaining contact lists, and deciding or suggesting whom to contact for different services. In this manner, agents help their members in finding the most useful and reliable parties to deal with, supporting some of the functionality of a registry in a distributed way. Referrals are based on a representation of how much the other available parties can be trusted. Agents are responsible to build and manage these representations taking into account the previous experiences of their members and collaborating each other. Participating on behalf of different members, agents appear as autonomous and heterogeneous. Moreover, agents organize themselves into communities and agents in the same community are called neighbours. Communities are dynamically formed according to the model that each agent maintains of some other agents. This model is based on the party's expertise (ability to provide correct services) and sociability (ability to produce accurate referrals). An example showing how a referral system works for online service selection is given in Figure 2.6. Agent **A** sends a query (i.e., a request of information about who provides a specific service) to its neighbors **B**, **C** and **D**. Agent **C** decides to ignore the request and, acting autonomously, it does not reply. Instead, agents **B** and **D** answer to **A**'s request but in two different ways. Indeed, in referral systems an answer can be a referral to another member (as in the **D**'s answer) or even oneself (**B**'s answer), in which case there would be some more interaction to actually provide the service. According to **D**'s referral, **A** decides to forward the query to **E** too. Again, **E** could reply with some referrals or proposing itself. **A** will take its final decision reasoning on the received answers.

**Definition 2.12 (Trust by Referrals)**
A service consumer trusts a service because of some referrals got from trusted software agents.

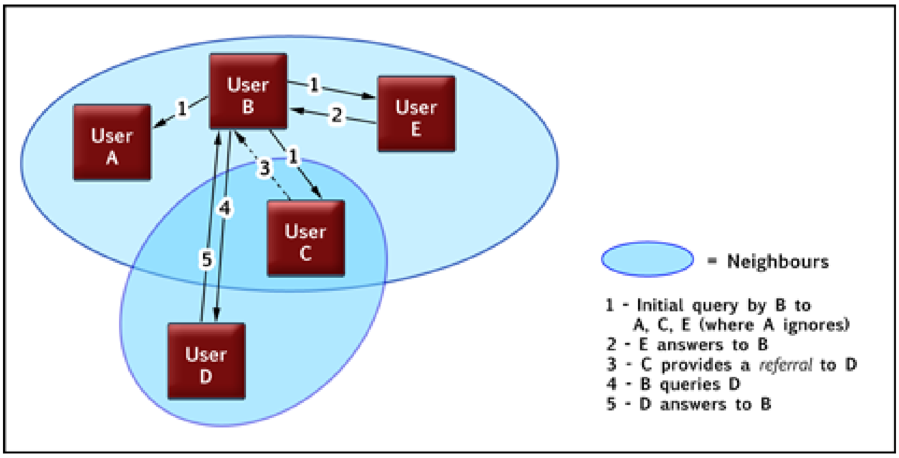


Figure 2.6: Example of Referral System for Online Service Selection.

Recommendation systems rely on a third party charged of computing recommendation. This authority does not necessarily have to expose the identity of the sources of the recommendations that it aggregates. On the contrary, in referral systems the participants reveal their ratings to those whom they trust, so the ratings would be more likely to be honest.

**Limitations:** Referral systems address some limitations of reputation and recommendation systems (such as, their centralized nature) but still rely on the judgements of the members of a community. Here the community is formed by software agents that act on behalf of their members (people, businesses etc.). Therefore, the effectiveness and practicability of the approach resides in the efficiency of the interacting community. Some technical practical issues, such as agents and members registration and communication as well as referrals representation, are left unanswered in the literature, making the impression of a still immature (or at least just academic) approach. Anyway, a part from these technical questions, the approach is still based on ratings coming from the direct previous experience of someone else, which leads to the main problem of selecting trustworthy services in the absence of some neighbors that can help us.

**Scenario 9:** Alice's discovery agents found a flight booking WS reference exploring a UDDI registry (for SOAP-based WSs). But the discovery agent does not belong to any community aware of that WS and then it cannot get any trustworthiness score.

#### 2.3.2.2 Matchmaking

These approaches are based on a component called *matchmaker*, responsible to match a user's request and trust preferences with available online service descriptions. If some matches are found than the results are sent back to the user. As shown in Fig. 2.7, two different matchmaking architectures have been proposed in literature, depending on the centralized or distributed nature of the matchmaker. A centralized trust-based matchmaking methodology has been

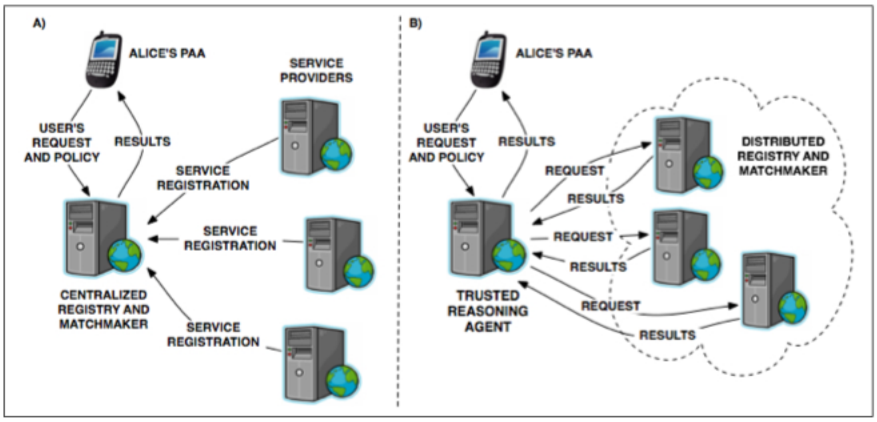


Figure 2.7: Example of Matchmaking Systems for Online Service Selection[24].

proposed by Galizia et al. in [28]. Differently to other approaches, it embodies the WS selection problem in a classification problem: given a set of user's policies and established a classification criterion, the goal is to identify a class of WSs matching the trust policies of the involved users. In other words, WSs are classified according to the specific user as well as trust policies. To do this, they have developed an ontology, namely WSs Trust Ontology (WSTO), that is able to represent generic trust specifications within the semantic WS-based interaction context. Being based on the WS Modelling Ontology (WSMO), WSTO can be supported by the IRS-III platform [22] which in this context behaves as a centralized trusted third-party by storing both user's profiles and services and reasoning on them (architecture A in Fig. 2.7).
A similar approach has been proposed by Olmedilla et al. in [70]. The main difference with respect to [28] lies in the underlying registry and matchmaking architecture, which is based on a P2P network (architecture B in Fig. 2.7). Whenever a new service provider wants to offer its services, it must just join such network. On the client side, a user looking for a service must send a query along with his policies to a reasoning agent he trusts. The agent distributes

the query to the peers on the network and each of them applies a matching algorithm. Whenever a peer has matches, it sends them back to the reasoning agent which joins the results and present them to the user.

**Definition 2.13 (Trust by Matchmaking)**
A service consumer trusts a service because a trusted (central/distributed) matchmaker states that the service's policy matches the consumer's request.

**Limitations:** As already discussed for reputation systems, a centralized architecture such as the trust-based matchmaking methodology proposed by [28] does not suit real open service oriented environments where the number of users and services might be very high. Having a single central matchmaker where both users and services must be registered to and where all the matches are computed is very far from the SOC vision. Moreover, to work correctly, the approach requires the users to disclose all their policies when they subscribe to the matchmaking system, since no trust negotiation is supported. This is in contrast with the openness of the system that would require a user to carefully disclose his/her policies. A consequence of this requirement is that the matching algorithm is not flexible and it is only based on a "take it or leave it" philosophy. Finally, it is not realistic to ask service providers to disclose all of their (maybe very sensitive) policies to a centralized registry.

Olmedilla et al. [70] replace the centralized matchmaker and registry with a Peer-to-Peer network, distributing the matchmaking process to the service providers. This would surely improve the performance and scalability of the matching algorithm, that might be computationally expensive to be executed on a single central server. Moreover, in a distributed approach servers can keep policies locally and private, which is an essential property in realistic open environments.

**Limitations:** The problem is moved from trusting a service or a service provider to the one of finding such a trusted reasoning agent.
Furthermore, the authors point out that *different groups of users might use different trusted agents, i.e., a university might set up an agent for its students and professors while a company could use a different one.* Relying on such computational entities, the approach is not appropriate for selecting trustworthy online services in SOC environments, because one cannot assume that these trusted agents will be always available for any context and any service.

**Scenario 10:** Alice is not willing to disclose all her trust policies to a central matchmaking service. Moreover, looking for a distributed matchmaking service she faces the problem of selecting and trusting a reasoning agent. But where can she locate such computational entities? And which agents should she trust? On which basis? Current distributed matchmaking technologies do not answer, leaving Alice in the same (vulnerable) situation.

**Limitations:** This family of approaches seems to solve the main problem of the community-based systems: the community-size-dependent quality. The trustworthiness rating is evaluated matching the user trust requirements directly against the WS provided trust guarantees. However this approach exposes another issue: in the real world, guarantees are not always met. It would not be difficult for a malicious (or distracted) user to craft a WS description so to pretend to be a trustworthy WS. But the *contract* might be fake and not respected by the provider. In this case a community would turn useful, providing evaluations based on the past experience.

**Scenario 11:** Let us assume that Alice submitted the company's trust profile to either a centralized or distributed matchmaker. Now a software agent is instructed to trust WSs providing the following trust guarantees:

- Encryption algorithm: AES
- Capacity: 20 simultaneous connections

The matchmaker essentially returns the matching WSs and Alice, by means of the agent, selects one. During the pre-easter week (when many people use the VTA system), after one month the VTA service is up and running, Alice finds out that the system is having many faults. Then she realizes that the *flight booking* WS cannot handle more than 10 connections simultaneously and the VTA system has to be temporary taken down.

### 2.3.3 Hybrid

Hybrid approaches for trust-based online service selection are based on a combination of well known trust methodologies, such as the ones addressed in previous sections. The key idea is that combining two or more approaches a consumer can improve the quality of the assessments.

#### 2.3.3.1 Socio-Cognitive

These approaches are mostly based on the works of Falcone and Castelfranchi [19][17][72]. Influenced by the Artificial Intelligence field and especially by the Multi-Agent System (MAS) paradigm, they treat trust as an agent's mental state. In this view, trust is articulated as an assumption or an expectation that a service consumer makes about a specific service. This expectation is based upon more specific beliefs which form the basis or the components of trust.
As pointed out in [7], beliefs can be seen as the answers to the question "What do we have in mind when we trust a service?". For example, we may trust a service because we believe that the service is able to do what we need (competence belief), and it will actually do it quickly (promptness belief). Competence and promptness are examples of such "mental ingredients", or beliefs, of trust. A belief describes therefore a state of the world from the point of view of an agent. That is, it represents the state the agent has in mind for a service: which/how is the agent's trust in (evaluation of) the service as for its competence and ability? Which/how is the agent's trust in (evaluation of) the service as for its intention and reliability? Which/how is the agent's trust in (evaluation of) the service as for its goodwill and honesty? And so on. According to this rationale, degrees of trust can be derived directly from the strength of the agent's trust componential and supporting beliefs. This leads to the following socio-cognitive definition of trust.

**Definition 2.14 (Socio-Cognitive Trust)**
The degree of trust is a function of the subjective certainty of the pertinent beliefs. Therefore, a service consumer trusts a service because of some of its subjective beliefs.

Examples of beliefs proposed in literature [19][17][72][7] include:

- Competence (or Reliability) Belief: the service's raw ability to accomplish a task, such as providing accurate results or performing a desired action
- Availability Belief: the availability of the service
- Promptness Belief: the speed at which the service responds to task requests by accomplishing the agreed upon task
- Cost Belief: cost refers to the monetary value that the consumer is willing to pay

Since the trust level is a function of such subjective beliefs, the approach requires the ability to form coherent beliefs about different characteristics of services and reasoning about these beliefs. A key question therefore arises: how are such beliefs obtained? That is, from which sources? The answer to the above question differentiates the various proposals in literature. The most common sources of belief can be summarized as follows:

- Direct Experience: the personal knowledge derived from participation or observation from direct interactions with a service (Section 2.3.1).
- Reputation: the global trust value resulting from the consumers' ratings of past interactions with the service (Section 2.3.2.1)
- Categorization: the process of grouping things based on prototypes (i.e., how the properties of a class are transferred to their members)
- Reasoning: the act of using reason to derive a conclusion from certain premises (i.e., more general than categorization)

For instance, in [43][82] the authors propose models in which they consider the direct interaction (experience) or reputation as sources. In [72] sources are categorization and reasoning. In [7] Ali et al. restrict sources to direct experience and reputation.

**Limitations:** A first weakness of the approach lies in the fact that it is based on beliefs obtained by means of the well know (and problematic) methodologies for trust. In other words, we are moving the problem of selecting a trustworthy service to the problem of selecting trustworthy beliefs that will be used as reasoning basis for deciding on the trustworthiness of the service. Another major limitation lies at the implementation level. To fully realize this approach, some sort of BDI[3] agents [75] is needed. Indeed, as Falcone et al. remarks in their paper [19] only a cognitive agent can "trust" another agent. We mean: only an agent endowed with goals and beliefs. This requirement seems too strong when applied to open and large service-based systems, since it is not reasonable to assume that every agent will be conforming to the BDI model (which, a part from the modeling of trust, requires specific architectures to support the reasoning on beliefs and goals).

**Scenario 12:** Alice is still searching for the most fitting flight booking WS to integrate to the VTA system. She is told then to delegate the task to an automated agent that will perform a reasoning process on the available WSs, interacting with other agents. Now Alice has to choose the agent that best suites her own subjective believes of trust.

[3]Belief-Desire-Intention

**Scenario 13:** Let us assume that a fitting agent is already available. This agent can manage to communicate with other agents of the same kind (BDI agents, for instance). Considering the openness of the system Alice is dealing with, where no strict rules are enforced, she realizes that her agent will not be capable of communicating with many other agents of the same kind. The selected WSs are then narrowed to a small amount compared to the total. The chosen WS is thus going to be far away from the best available choice.

#### 2.3.3.2 Trust & Reputation

Studies such as [39][40][45] propose methods for assessing the quality of online services by combining trust and reputation techniques in a single integrated framework. For instance, [45] shows how (Bayesian) reputation systems can be combined with trust modeling based on subjective logic [44]. [39] describes a report-driven framework. In this study, WSs have their QoS profile computed by means of reports provided by both providers and consumers. The profiles are used to generate a WS rank based on the consumer requirements. This is the reputation side of the framework. On the other hand, it is necessary to guarantee the reliability of the provided report, because a real world scenario would also include both liars and distracted users that would poison the ranking system with false/wrong reports. Thus the authors suggest, along with the reputation framework, a system (enhanced by some previously trusted entities) capable of identifying *liars*.
The rationale of these approaches is that by combining two different trust methodologies the resulting integrated framework will improve some weaknesses of the constituent methodologies, and thus the overall assessment of online services.

**Definition 2.15 (Trust & Reputation based system)**
A system providing for a trustworthiness score employing methodologies based on both reputation and trust. One is used to enhance the result supplied by the other.

**Limitations:** Although these approaches are remarkable, especially [45] where the integration results in a flexible framework for online trust management, they still suffer the main limitations of their constituent methodologies. For instance, both approaches inherit one of the main weaknesses of reputation systems, that is to be based on a centralized and trusted reputation center (Section 2.3.2.1). Moreover, as for [45], there are many issues inherited by the community-based trust provisioning systems that have not been addressed yet.
The authors of [45] propose a bootstrapping method consisting of creating

trusted reports for the most important WSs by means of trusted monitoring agents. With this approach there would be the problem of selecting the *most important WSs* nonetheless.

**Scenario 14:** Alice's WS discovery agent obtains a rank of the most fitting flight booking WSs. Alice now knows what the centralized system "think" about the available WSs, but she does not know whether the data used for the computation were enough to generate a useful rank. She might has to wait for a long period before having a good evaluation of the available flight booking WS.

#### 2.3.3.3 Direct Experience & Reputation

[80][73][12] propose a model where the trust in a service is computed as a rating of the level of performance of the service. This overall performance is not limited to the agent's direct experience (or confidence, see Section 2.3.1) but it also based on the evaluations by other agents in the system (in [80] it is named the *group experience*, i.e. what the other members of the group think about the agent being evaluated and of his group). Thus, in these models, trust can be seen as a rating built as a result from combining agent's direct experience (with the service) along with the social reputation of the service provider.

**Definition 2.16 (Trust by Direct Experience & Reputation)**
The trust towards a service is evaluated by means of the user direct experience combined with the service reputation.

**Limitations:** Again, the combination of two methodologies improve some weaknesses of one constituent model, but it does not provide a complete solution to the trustworthy online service selection problem. For instance, in [73] the authors combine confidence and reputation to address the situation where no previous experience of the service is available (main weakness of the direct experience method). But to do this they based their proposal on trust and reputation mechanisms to infer expectations of future providers' behavior from past experiences in similar situations. This idea inherits the already discussed problems of trust and reputation mechanisms.

**Scenario 15:** Alice's agent cannot establish a reliable trustworthiness score for certain flight booking WSs because there are no past interactions with them and, moreover, they seem to have joined the WS network too recently in order to have some useful reputation evaluations.

### 2.3.4 Automated Trust Negotiation

Automated Trust Negotiation (TN) [11] is an approach specifically targeted to allow agents to access sensitive data and services in open environments. Trust negotiation protocols are based on the iterative disclosure of digital credentials and requests for credentials between two unknown parties (strangers in TN jargon), with the goal of establishing sufficient mutual trust so that the parties can complete a transaction. Informally, digital credentials (credentials for short) refer to the online analogues of paper credentials (a drivers license, passport, or employee ID card, for example). Thus, a credential is a digitally signed assertion by a credential issuer about the credential owner. It is usually signed using the issuers private key and verified using the issuers public key [94]. To automate trust negotiation, each party must establish access control policies (policies for short) to protect its sensitive resources, including credentials and services, from inappropriate access. Each policy should specify the digital credentials strangers must present to access the protected resource. Policies can themselves be seen as sensitive resources. To better understand how TN systems work, let us apply the approach to our Virtual Tourism Agency scenario, under a different point of view: Alice in this case is not a developer, but an occasional user that has to plan a trip helped by a VTA software agent installed in her smartphone.

**Scenario 16:** To trust the discovered *flight booking* service, Alice's discovery agent decides to start a Trust Negotiation with the *flight booking* service provider, as shown in Figure 2.8. The steps of the negotiation are described below.

1. The negotiation starts with the agent request of the service. At the time of the request, the agent has no prior knowledge of the *flight booking* WS provider's requirements for granting access to the service.

2. The *flight booking* WS provider replies sending a policy to the agent. Such policy specifies that the agent must submit two credentials, namely Alice's passport ID and VISA credit card number, in order to use the agent service and to make an online booking.

3. Alice wants to disclose confidential information only to trusted third parties. Thus the agent is programmed to unconditionally disclose the client's passport ID for the *flight booking* service, but it is programmed to disclose

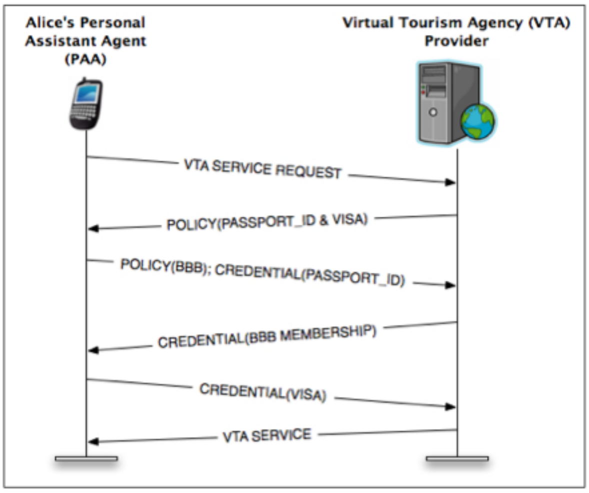

Alice's Personal Assistant Agent (PAA)
Virtual Tourism Agency (VTA) Provider
VTA SERVICE REQUEST
POLICY(PASSPORT_ID & VISA)
POLICY(BBB); CREDENTIAL(PASSPORT_ID)
CREDENTIAL(BBB MEMBERSHIP)
CREDENTIAL(VISA)
VTA SERVICE

Figure 2.8: Example of Trust Negotiation for Trust-Based Online Service Selection[24].

VISA card number only to a business that is a member of the Better Business Bureau (BBB). Therefore, the agent replies to the *flight booking* service provider disclosing the client's passport ID and requesting a BBB membership credential.

4. Fortunately, the *flight booking* service provider is a BBB member and discloses the membership number to the agent.

5. Having the policy satisfied, the PAA replies disclosing client's VISA card number to the *flight booking* service provider.

6. Finally, the *flight booking* service provider allows Alice's agent to invoke the service and the TN protocol ends successfully. At the end of the negotiation, Alice's agent trusts the *flight booking* WS provider and the *flight booking* WS provider trusts the agent.

The above example should make clear the concept of trust in TN systems. Here the point of view is not restricted to the service consumer only (how the service consumer may trust a service) but the goal is to establish a mutual trust between service consumer and provider.

**Definition 2.17 (Credential-Based Trust (or Trust by Negotiation))**
A service consumer and a service provider mutually trust each other because

the access control policy of the requested service is compliant with the access control policy of the service consumer.

Note that the above definition does not state that a negotiation will always succeed if the parties' policies are compliant. Indeed, the success of the negotiation depends on several factors. For instance, a negotiation could take different routes according to the negotiation strategies adopted by the parties [49]. The above definition just states that if a trust negotiation succeeds establishing a mutual trust among two parties then this is because the two parties have compliant access control policies for the requested resource.

**Limitations:** Trust negotiation principles and systems have been widely investigated in the last few years, both in different (still mainly academic) domains (like e-Business, e-Commerce, P2P systems and more recently in WSs [25]) and with respect to issues such as privacy, safety and efficiency. This effort is evident in the growing literature on TN related issues ([94] [101] [99] [56] [69] [70] [57] [11] [86] [87] to mention only a few). However, several key issues have still to be addressed to bring Trust Negotiation to its full potential. Listing all these weaknesses is outside the scope of the paper and in the following we will try to identify only the ones that are relevant to the online service selection problem.

**Lack of Real-World TN Systems:** to date, research in Trust Negotiation has been primarily of a theoretical and academic nature, resulting in a strong theoretical foundation of the matter but developing only few proof of concept prototypes ([94][3][14]). Real-world Trust Negotiation systems are still missing.

**Lack of TN Standards:** to implement a real-world TN system, a number of important technology-related issues must be addressed and to date no standards have been identified. For instance, one can find many languages for expressing resource access policies (e.g., [59][10][2]), several protocols and strategies for conducting trust negotiation (e.g.,[101][3][50][25]) and different logics for reasoning about the outcomes of these negotiations (e.g., [15][95]). As a result, proof of concept implementations are based on different languages and protocols, making the different systems unable to talk each other.

**Tailored to Credentials and Negotiation:** Even assuming that some standards will be eventually defined, exploiting a TN approach for selecting online services would require that both parties (client and service provider) should be able to support a (complex) negotiation process. This sounds a too strong requirement for open large systems, where consumers should be able to select a trustworthy service with less computational effort and not necessarily after a (complex) negotiation. Moreover, adopting a TN approach would require that both parties reason and act according to a credential-based notion of trust. Other trust meanings are not supported.

**Tailored to ingle Service:** current TN approaches take for granted that a client always starts the negotiation by requesting access to a resource. Instead, as pointed out by Mecella et al. [67]:

> *While many in the literature treated Web Services as a set of independent single operations, interacting with real world Web Services involves generally a sequence of invocations of several of their operations, referred to as conversation. A simple example is a bookstore Web Service; buying a book involves generally searching for the book, browsing the details and reviews about this book, adding the book to the shopping cart, checking out, paying, etc.*

It is therefore of key importance to consider the access control and negotiation issues for the overall WS conversation. As noted by Koshutanski and Massacci [49] it might well be that the conversation takes different routes, therefore changing the set of needed credentials. Keeping up with the book example, if we decided to send the books as gift then we only need to specify the address of the credit card holder and the address of the gift recipient. Our address is not needed. Mecella et al. [67] have provided an access control model and a trust-negotiation scheme for WS where such conversational aspect is taken care of. While they take full care of the behavioral aspect of WS, their negotiation protocol still sticks to the progressive disclosure of credentials while keeping the set of requested services fixed.

**Heavy procedure with many available WSs:** in order to kick off the negotiation, the user has to establish a communication directly with the provider. In case the user discovered a considerable number of WSs where he has to pick-up one from, he should instantiate a negotiation with each provider in order to find out the trustworthy one. This can result in an additional workload for service providers and, moreover, in a time-consuming and resource-consuming process (for the user). This might not result effective in some scenarios, such those where a user is consuming WSs with his smartphone (sometimes a not really powerful device).

**Some SOA use-cases unawareness:** in a SOA there are use-case where the services are supposed to be consumed many times from the same costumer, since they are involved in a choreography. This means that, for instance, a WS should be trusted the first time during discovery (development) and then be consumed without the TN protocol being involved on the ensuing requests. Once the trust is established, there should be a way to cache it for a period. We are usually willing to trade off disclosure of our security attributes for each *additional* or *unknown* service.

**Only contract level guarantees:** A service consumer can trust a service provider by means of this incremental policies disclosure until an access level is agreed between the two parties. But what if a service provider claims (for instance) a false scalability value among its QoS contract parameters of his pay-per-month WS? In such a case it may happen that after one week of usage the service stops working adequately. That would require that someone who experienced this problem is able to make other future consumers know about the issue.

**Scenario 17:** Back to our scenario, Alice might not be willing to disclose her credit card number for the *flight booking* service. But it might be willing to do so if the *flight booking* WS provider tells her that the service gets a 10% more quota if the credit card is disclosed. But unfortunately, current TN frameworks do not allow to negotiate services for trust and viceversa. A first preliminary work on this direction has been proposed by Dragoni et al. in [25][26].

### 2.3.5 Discussion

#### 2.3.5.1 Main Issues

As we verified throughout this analysis, many approaches have been suggested to enhance the service selection process with trust. Some of them are not directly addressed to a SOA environment, but can still be taken into account for further improvements and extensions in order to adapt them to SOC. The literature about that is growing, but, as it is possible to verify out of this analysis, WS trustworthiness provisioning is still an open challenge.
While investigating on studies in literature, it has been possible to derive shortcomings and advantages of the single approaches. They are summarized in the table 2.1. In order to optimize space and improve clarity, the plus and minuses are synthesized in few main classes, each one with its notation:

**Shortcomings**

- **NSR** (*New Service Rump-up*): refer to Def. 2.6;

- **CD** (*Community Dependent*): a community dependent system is affected by the community bootstrap issue (Def. 2.7);
- **NUR** (*New User Rump-up*): this problem is connected to those approaches where it is necessary to calculate the user profile in order to provide some useful evaluation to him. This means that the user has either to use the system for a period (Def. 2.11) or to provide some sensitive information before having valuable trust scores.
- **HS** (*Hard Setup*): this problem is connected to those approaches that require a big effort to be integrated in the real world;
- **UT** (*Unconditional Trust/Distrust*): this issue is related to the approaches were the user has to consume a service without any previous experience or evidence that the service is trustworthy. Or, the other way around, the user distrust the service unconditionally for the same reason;
- **CE** (*Centralized*): deriving from the centralized nature of the given approach (Def. 2.2):
  - black-box score computation
  - single point of failure
  - usually need to disclose sensitive information to a central entity
  - not fitting to SOA because of its intrinsic scalability demand

**Advantages**

- **PUTS** (*Pre Use Trust Score*): chances to obtain a trust score *before* using the service;
- **UFS** (*User Fitting Score*): the WS trust score is also somehow related to the user personal "tastes" and habits;

Apart from the features listed above, there are some other specific pros and cons related to certain approaches that will be described directly in the table.

| Approach | | | Pluses | Minuses |
|---|---|---|---|---|
| **Direct experience** | | | UFS $\rightarrow$ the **most** fitting score | UT |
| **TTP** | Social | Reputation | PUTS | NSR, CD, CE $\rightarrow$ for those methodologies based on a centralized architecture |
| | | Recommendation | PUTS, UFS | CD, NSR, NUR or user information disclosure, CE $\rightarrow$ for those methodologies based on a centralized architecture |
| | | Referrals | PUTS, rates coming from trusted peers | NSR, CD |
| | Matchmaker | | PUTS, UFS, some community based methodologies provide for *liars* recognition | HS, CE $\rightarrow$ for those methodologies based on a centralized architecture, trust towards service moved to trust towards agent |
| **Hybrid** | Socio-Cognitive | | Accurate trust computation, UFS | depending on the belief source $\rightarrow$ UT/NSR/CD/CE/NUR, HS $\rightarrow$ cognitive agents have to be conforming to a model, |
| | Trust & Reputation | | *liars* recognition (some of the ideas), PUTS, some sort of good results can be provided even with a poor community or a brand new service | CE, NSR, CD |
| | DE & Reputation | | Issues of the 2 constituent models mitigated | NSR, CD |
| **Automated Trust Negotiation** | | | UFS, PUTS, trust can ALWAYS be computed | HS, no standards defined, no WS aware (at the current state) |

Table 2.1: Pluses and Minuses summary

In order to carry out this evaluation, few questions have been adopted as guidelines to judge each system:
***How does the trust score fit the user needs?*** It would be better to build the trust score around the user profile, as, for instance, the recommendation

systems do;
***Does the provider/consumer have to disclose any sensitive information?*** Some centralized approaches ask the user (be it either the provider, the consumer or both) to submit some personal information in order to improve the trust score computation. This is clearly something that should be avoided: users usually do not want to unconditionally disclose sensitive details to a central authority.
***Can the user know how the trust is calculated?*** Depending on the system architecture, the trust score might be calculated by a third party in a black box. The user may rather prefer to know how the service he is going to trust has been suggested.
***How does the community influence the trust score?*** This issue mainly affects the *social* approaches (2.3.2.1): is the trust score depending on the size/quality of a community? In such a case, the main shortcoming would be the community bootstrap, i.e. how to create an initial community to kick-off the system.
***Does the user has to unconditionally trust/distrust certain services?*** Whenever a user finds a WS, if the functional contract meets the user needs, there should always be a way to provide a trust score for that service, without leaving the user in the position of unconditionally trusting/distrusting the WS.
***What is the trustworthiness of a brand new WS?*** A new WS (i.e. it has been recently deployed) needs a way to be "tried" even with no previous knowledge about it. A non functional contract or TN approaches seem to be a good starting point to address this issue.
***How hard is the trust provisioning infrastructure to setup and maintain?*** When designing a trust provisioning system it has to be taken into account both the effort needed to apply the system to the already existing SOC infrastructure, and the issues intrinsically related to the nature of a SOA-based system. A trust provisioning system is in charge of a great responsibility, and its robustness and scalability is a critical point in a SOA environment. Thus, for instance, it would be rather irresponsible to adopt a pure centralized architecture (that would be the single point of failure).

#### 2.3.5.2 Soft Trust VS Hard Trust

Rasmusson et al., in [76], coined the terms *soft security* and *hard security* to categorize security mechanisms. Following the same idea, we further narrowed down all the classes of approaches in two main families: those based on a **soft** notion of trust and those based on a **hard** one. The first one comprises the vast majority of the approaches. More precisely, those that draw the trustworthiness of a service out of the following sources:

- direct experience of the consumer with the service
- indirect experience (opinions on the service coming from someone trusted by the consumer)
- some form of combination of direct and indirect experience (what we called hybrid approaches)

All of them has two key limitations in common: the user has to obtain *trust* from his own direct experience *or* from the direct experience of someone else he trusts. The former is affected by the problem of the blind execution. In the latter, the same problem is just moved to another level: someone else has to blindly use a service. This requires 15 people taking the risk to *try*. As correctly pointed out by Dragoni in [23]

> *if someone does not take the risk of invoking an unknown service for the first time, then no one will be able to decide about the trustworthiness of the service before its invocation.*

Approaches based on the *soft notion* of trust share the critical issue of new service and community start-up (Defs. 2.6 and 2.7).
The rationale of the soft trust is that participants in a market collaborate each other in sharing information on other participants or services. Soft trust expects and even accepts that there might be malicious services or service providers in the system. The idea is to identify them and prevent them from harming the other participants by means of collaboration and social interactions, aiming at sharing as much knowledge as possible.

**Definition 2.18 (Soft Trust)**
Soft trust is a user's belief of a service trustworthiness based on a social control philosophy. It demands his experience or the past experience of a society with the service. This allows malicious users to be identified and put aside. All the community dependent approaches are based on a soft notion of trust.

The other class of approaches, such like Trust Negotiation and Matchmakers, relies on a *hard* notion of *trust*. The rationale is that it should be possible to derive the trustworthiness of a WS just looking at its non functional contract (like QoS information). Some of them take into account the *semantic* of a WS, i.e. services should be selected considering their security behaviour (for instance, access control rules). The recent Security-By-Contract (SxC) approach illustrated in [25] might represent a good starting point for this purpose, because it takes into account the security behavior of a service instead of depending on

the social control philosophy in the existing trust based approaches. Moreover, it is directly conceived considering the SOA related issues.
Nevertheless, even the *hard trust* provisioning approaches studied in literature have a critical drawback: the lack of lying/distracted user recognition capabilities. In other words, everyone can provide a fake/wrong contract, be it due to either a malicious behaviour or human distraction (or other unpredictable problems). In this case the community help would turn useful.

**Definition 2.19 (Hard Trust)**
Hard trust is a user's belief of a service trustworthiness based on a guarantee that the service will be trustworthy. It demands the user to be certain the service will behave as stated.

Moreover some practical issues related to WSs are not yet considered. For instance, the Trust Negotiation approaches work fine assuming that the WS is consumed by a user directly invoking it. But in the real world, as pointed out in [68], WSs can even be a "developer thing". This means that they have to be trusted during the development time and then transparently consumed by clients unaware of the distributed nature of the system they are using.

#### 2.3.5.3 Conclusion

The analysis carried out throughout this chapter constitutes the jumping step for the design of a new trust provisioning framework conceived for SOA. The issues arisen during the *discussion* and the considerations regarding the *hard and soft* notions of trust are the guidelines that will drive the creation of a new approach.
Furthermore, as we pointed out in the introduction of the analysis, there is a sort of term confusion related to words "trust" and "trustworthiness". A clear distinction of their acceptation is still missing in the currently suggested approaches. Thus, an agreed definition on their meaning is necessary to provide the cornerstone for a new framework.

CHAPTER 3

# Framework

In this Chapter we are going to describe the framework conceived to meet the requirements listed during the preliminary analysis in Chapter 2.
In the introductory section the founding principles that constitute the rationale of our work are explained. Essentially, the framework will be firstly described in its high level architecture and then detailed to its components.
The framework has been designed trying to follow the DRY[1] principle, i.e. trying to adopt already existing ideas and focusing the effort on integrating them. Some of them needed a further extension or revision to well fit our system. This choice derives from the realization that completely redesigning new solutions in this case would signify to reinvent the wheel, considering the vast amount of studies on this matter already present in literature.
As a final clarification, it is necessary to point out that the framework specification has been willingly kept abstract enough to leave free choice on its future application (no matter if it will be applied to SOAP based Web Services, REST-ful ones or any others). Thus, details related to technologies, languages and so on are left to the final chapter where some test cases and implementative examples will be illustrated.

[1]Don't Repeat Yourself

## 3.1 Founding principles

### 3.1.1 Soft and Hard Trust Integration

The analysis in Section 2 highlighted the separation between soft and hard notions of trust, focusing on the advantages and weaknesses of them. It then emerged how hybrid systems generally turned to improve constituent methodologies. The joint collaboration of more techniques usually blunts the shortcomings of the single approaches. These considerations suggested one of the founding principle of our framework: **hard trust** and **soft trust** provisioning techniques should be embodied in a unified hybrid model. The user will then always be able to evaluate *automatically* the available WSs for a given need, while the community will push unworthy WSs aside. In other words, the user will *always* be able to make up his trust belief towards a WS and malicious users/services bypassing the hard trust mechanism of the system will be caught by the community.

### 3.1.2 Trust vs Trustworthiness

Understanding the meaning of the term *trustworthiness* represents a fundamental step towards the definition of a theory of service trustworthiness. Indeed, as we analyzed in Chapter 2, the literature shows that a common agreement on its precise meaning is still missing, leading to a confusing situation where terms such as "trusted" and "trustworthy" are often interchanged, and the term "trustworthiness" is overused without any precise definition of its meaning. An illustrative example of such ambiguity is considering the definitions of trust and trustworthiness given by Bishop and Chang et al. in [13][20]:

**Definition 3.1 (Bishop)**
An entity is *trustworthy* if there is sufficient credible evidence leading one to believe that the system will meet a set of given requirements. *Trust* is a measure of trustworthiness, relying on the evidence provided.

**Definition 3.2 (Chang et al.)**
*Trust* is the belief the trusting agent has in the trusted agent's willingness and capability to deliver a mutually agreed service in a given context and in a given time slot. *Trustworthiness* is a measure of the level of trust that the trusting agent has in the trusted agent.

Analyzing all the differences between these definitions is outside the scope of our investigation. However, what should become apparent at a first sight is that there is a strict correlation between trust and trustworthiness, although such concepts are defined in different ways. For instance, by Def. 3.1, an agent trusts or not a service according to the service's trustworthiness, so trust relies on trustworthiness. By Def. 3.2 instead trustworthiness relies on trust. So in the former definition trust is seen as a measure of trustworthiness, while in the latter it is exactly the contrary, i.e. trustworthiness is defined as a measure of the level of trust.
Some interesting questions therefore naturally arise: what is the nature of trust and trustworthiness? Do they refer to the same concept? Or are they distinct? And in that case are they correlated? And how? A comprehensive philosophical answer to these questions is outside the scope of this work. In order to proceed with the explanation of the framework it is necessary to agree on a definition of what we will mean throughout the document with the terms *trust* and *trustworthiness*. Loosely speaking, trustworthiness implies that something is worthy of being trusted. We see it as a subjective property, that relies on the user's point of view to some extent. Trust merely implies that one trusts something whether it is trustworthy or not, perhaps because one has no alternative, or because one does not even realize that trustworthiness is necessary, or because of some other reason. This distinction is expressed very clearly in the following quotation:

> *Trust is an attitude that we have towards people whom we hope will be trustworthy, where trustworthiness is a property, not an attitude. Trust and trustworthiness are therefore distinct although, ideally, those whom we trust will be trustworthy, and those who are trustworthy will be trusted.*[66]

According to these considerations, we aim at providing an informal definition of *service trustworthiness* and *trust* which will constitute the fundamental premise of the suggested framework and will hold henceforth in the document:

**Definition 3.3 (Service Trustworthiness)**
A service is trustworthy if there is sufficient credible evidence leading a user to believe that the service will meet a set of given requirements.

Thus, the framework must support three key concepts:

- a *set of given requirements* the the service must meet during its execution;
- a *credible evidence* that can be checked by the user to determine trustworthiness of the service; in other words, the evidence is *credible* because it can be verified (or monitored) by the user;

- a notion of *sufficient evidence*, that is a “quantity of evidence” that a user needs for believing on the service trustworthiness.

The definition of *trust* a user has towards a service is:

**Definition 3.4 (Trust towards a Service)**
The user trusts a service if he has a strong belief that the service will meet a set of given requirements, not necessarily rationally.

The *strong belief* is then up to the user himself, who has free will in the choice of whether to trust a service or not. This means that, even with many trustworthiness evidences, the user might decide to not trust it (so, in such a case, the service is not trustworthy for the user).

This agreed definitions of service *trust trustworthiness* are going to constitute one of the founding principles of our framework.

## 3.2 High Level Architecture

The architecture of the framework is conceived to support the founding principles explained in 3.1. According to Def. 3.3, the concepts of *sufficient evidence* have to be supported. A user willing to consume a service needs some evidences that the service will be trustworthy. It is then up to the user himself how to evaluate and weigh these evidences. The framework has then to provide the tools helping the potential consumer to take a choice, without forcing a unique *perception* of trust.

A further key-concept the framework has to support regards the *set of given requirements* the service *must* meet during its execution. In order to make sure a service is meeting a set of requirements, a *contract* is necessary. A *negotiator* must then agree on a contract with the provider. The contract will then be monitored to make sure the provider respects it.
The monitoring process will provide what we defined the *credible evidence* that the service is behaving well, i.e., as claimed in the agreement. Note that the evidence concerns the fact that the service has satisfied the agreement in a conversation.

Another cornerstone forming the rationale of our framework is what during the state of the art analysis we defined as *hard trust* and *soft trust*. The former is a belief based on the existence of a non functional contract, a guarantee that the service will behave as requested. The latter is instead based on the evaluation of a society, capable of recognizing liars or who generally cheat (wittingly or unconsciously) the *hard trust* of the users. As stated in 3.1.1, the evidences the framework has to supply should let the user decide based on both a hard and a soft belief.
Finally, there have been listed many different issues that should be solved. Some of them affect more approaches in the same time, some others are specific to certain solutions. The framework has been conceived to overcome even these issues. The discussion in 4.2 will explain how and where.
Fig. 3.1 shows the high-level architecture of the framework, leaving the most part of the details aside (they will be explained in the following sections).

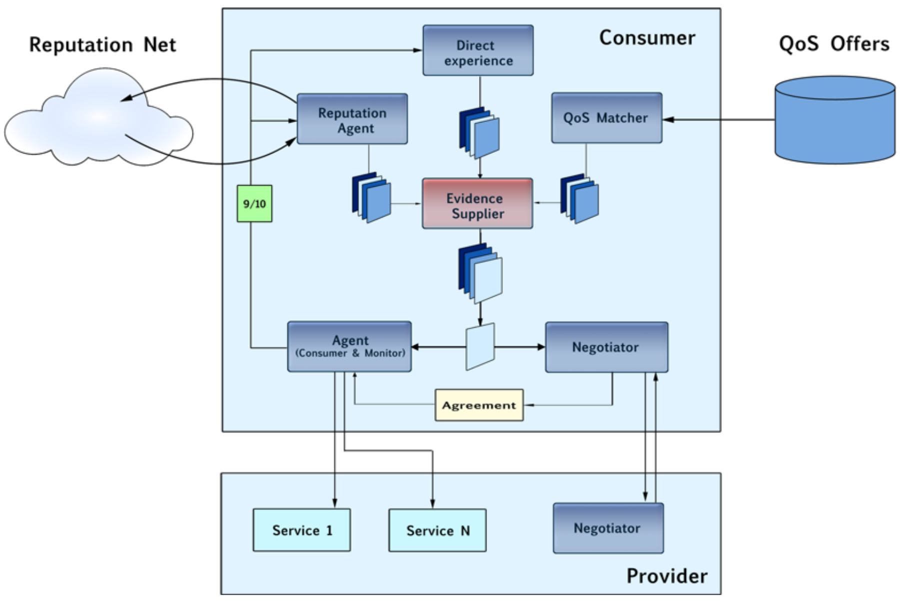


Figure 3.1: Framework - High Level Architecure.

The evidence supplier acts as an *aggregator & ranker* plugged to different sources to draw on for providing evidences to the user. Each source is fed with the list of (functionally similar) WS references that will then be ranked depending on the technique applied by the source. The evidence supplier will then: (1) normalize the scores of different sources to the same range (min-max normalization), (2) aggregate them according to the user supplied weights, (3) re-rank the resulting list, (4) filter it according to the user supplied thresholds. For instance, a user may decide that source **A** is more important than **B**, assigning 80% of importance to **A** and 20% to **B** and "unplug" source **C** by giving a 0% weight. The resulting ranked list can then be filtered by specifying that the services below 10% score in the rank are automatically not trustworthy. It is, essentially, up to the user the choice of what should be the **sufficient evidence** he needs for believing the service trustworthiness.
A *negotiator* is then in charge to agree directly with the *provider* on a *contract* that will be *monitored* throughout the whole interaction between consumer and provider. This contract constitute the **set of requirements that the service must meet during its execution** and the monitoring process will help the automatic evaluation of the consumer experience over the time.

A point is still missing: by which means *hard trust* and *soft trust* are provided?

**Soft trust sources:** as mentioned in Chapter 2, soft trust derives from direct experience, being it by the consumers himself or by someone else's experience. In the framework, we considered then two soft trust sources: a reputation community and the user direct experience.
The **reputation** based source is the community that an agent installed in the client will contact to obtain the reputation scores for a given list of WSs. This is one of the sources the *evidence supplier* uses to provide a static score for a WS trustworthiness.
Furthermore, the direct experience with a WS is also used as a *soft trust* source. This source can clearly have a role just when the user already had some sort of interaction with a WS. We provided the tool to let the user automatically feed this source: the negotiator along with a monitor, better explained in section 3.6.

**Hard trust sources:** these sources are based on guarantees given by the provider. The first one is based on the QoS offer of the provider. We will explain in details the motivation behind this choice in 3.4. This is another source used by the evidence supplier to help the user making up his own idea about a WS trustworthiness. Then, in order to leave the providers specifying "private" policies, a contract by negotiation procedure is provided. This way, the user will be able to negotiate specific (and different from QoS) requirement for the ensuing conversation with the WS and have a *guarantee* on the future service behaviour that he will be able to monitor.

The preliminary categorization performed by the evidence supplier helps the user to avoid a random selection for a trust negotiation procedure. This solves the problem explained in 2.3.4: the user does not have to instantiate a connection with every provider before being able to take a decision. A preliminary suggestion will be supplied by the evidence supplier.

As already stated in the introduction of this chapter, in our framework we tried to *outsource the components* when possible, or, in other words, we tried to drawn the techniques for trust provisioning from other studies. This choice has been taken because many studies have already addressed the trust provisioning issue and we believe some of them can be reused and directly integrated in this system.
Before providing more details about the framework, let us point out 2 premises:

1. It is assumed that the functional WS discovery has already been performed: it would be out of the scope of this work to provide details concerning the WS discovery. We assume the WS contracts that match the functional requirements of the service consumer have already been found and are now available to the user;

2. The *QoS Offers* of the WS providers are owned by an external entity in charge of collecting them. We assume that such an entity must exist, but we leave the design/implementation details to other researches, because the topic is not directly related to trust provisioning for SOA. It has been inserted in the figure for clarity reasons and it will not be analyzed in this document.
   Nevertheless, we advocate that the non-functional contract of each WS *should* not be retrieved by the client/agent from the server itself. There are various motivations, some of them already stated in [98]: keeping a local copy of a QoS contract would introduce some concerns regarding when the client should update it. Assuming the existence of many, frequently updating agents, the WS provider would have a considerable additional workload. Moreover, the process of querying one by one all the web service providers matching the functional contract in order to obtain the QoS contract, would considerably slow down the selection process.

### 3.2.1 Roles

The entities and components listed in the previous section collaborate in order to provide to the user a *sufficient credible evidence* of the service trustworthiness. They will try to make the user trust a service (refer to definitions 3.3 and 3.4 in section 3.1.2) or not, without leaving him in a vulnerable position of taking a random choice. Below the components' roles in the system are listed:

1. The *QoS Matcher* matches the user's *qos demand* against the *QoS Offers*. It then discards those offers that do not meet the consumer requirements and rank the remaining contracts by "fitness" with the consumer demand. The category this component is mapped to is the **matchmaker** one. Nevertheless, the matching procedure is applied client side, in order to keep it decentralized and overcome some the issues listed during the analysis.

2. The *Reputation Agent* communicates with its peers in the reputation network to obtain a reputation score for each service. The agents in the reputation network communicate each others as well in order to have a valuable reputation score for a service. At the end the reputation agent rank the WS references based on the reputation score.
   As we will see in more detail later, this component is designed along the lines of the **referral** systems, providing the user with the same service as **reputation** services do.

3. The *Direct experience* is not probably available at the very beginning. It is built after the user already consumed a WS. It is calculated based on

the WS behaviour over the time with the help of a monitoring agent.

4. The *Evidence Supplier* filter and rank the WS references based on the result provided by the different *sources* and applying some user-defined thresholds and weights.

5. The *Negotiator*, once a WS is either automatically or manually selected, will negotiate with the provider an agreement based on the user policies. The negotiator component has been inspired by the **trust negotiation** methodologies.
   During the negotiation, the security behaviour of the service will be agreed. Further details will be explained in specific sections.

6. An *Agreement* is produced out of the negotiation. This is used as a contract by a monitoring agent to automatically evaluate the service behaviour during the conversation with the provider and to release ratings to be used as reputation scores or direct experience scores.

Depending on different factors, some of the aforementioned components might not be used, without anyway affecting the whole procedure. These factors can be related to the user willing to do not use a specific source for the evaluation. Or it can even depend on the unavailability of certain infrastructure components (e.g. the provider does not have a negotiator agent).

## 3.3 Evidence supplier

The definition provided for *trustworthiness* in 3.1.2 relies on the concept *sufficient evidence*. It results clear the relation between the *trustworthiness* and the possibility of providing evidences about it. However a user has his own belief of what should be trustworthy (and thus, he personally decides whether to trust a service or not). This means that, depending on the user's attitude, a service may be trusted even with either a really weak evidence (the user is like a "gambler") or, the other way around, a very strong evidence (the user is "paranoid").
The **evidence supplier** (ES) is then in charge of providing what we defined the *sufficient evidence* of a WS by collecting information from different sources. The definition of *evidence source* in this scope is:

**Definition 3.5 (Trustworthiness evidence source)** A function, component, methodology or any entity capable of producing an evidence of trustworthiness for a given service.

As already mentioned in Section 3.1.1, we believe that an evidence of trustworthiness should let the user rely on both *hard* and *soft* notions of trust. Thus, the *evidence supplier* should collect enough information from other trust provisioning mechanisms so to make the user take his choice.
We will discuss about the sources that are going to be integrated in the system in Sections 2.3.2.1 and 3.4.
The effort has been to devise a methodology to aggregate different evidence sources, letting to the user the flexibility to model the result according to his trust needs.

### 3.3.1 ES/Sources collaboration

#### 3.3.1.1 Sources

The *evidence supplier* will obtain trustworthiness information from three sources:

- A *QoS Matchmaker* component will filter and rank a given list of functionally matching WSs based on the user's QoS demand and the services QoS offers.
- A *Reputation* component will filter and rank a given list of functionally matching WSs based on the community's reputation towards the service.
- The *Direct experience* component keeps track of the past experience of the user himself towards the services. It is computed by automatically monitoring the activities with a WS.

The ES will have to filter & rank the WSs based on the result returned by the different sources. This implies that the results should respect a defined structure in order to be aggregated. Thus, every source should return a result following the listed requirements:

- The evaluation of the WSs has to be expressed by means of a ranked list, where each entry is a tuple containing the WS reference and the given score;
- The rank has to be monotonically increasing, i.e. the higher the score, the higher the rank;
- WSs that have not been evaluated for information unavailability (e.g. no QoS offers), has to be returned in a separate list of *unclassified* results.

#### 3.3.1.2 Scores normalization

In order to decouple as much as possible the ES from the different evidence sources, the task of normalizing the scores to the same scale is delegated to the evidence supplier. This way, the engine of each source does not have to be designed in order to respect a specific scale that might change over time.
To fulfill the requirement, the **MAX-min normalization** has been used. This choice is based on the consideration that different evidence sources may return ranks in different ranges of values. With the MAX-min normalization it is possible to narrow down all the values in the range [0,1], intuitive to understand.

#### 3.3.1.3 Results

The *evidence supplier* is fed with a *given* set of WS references. We then assume (as already stated in Section 3.2) that a functional matching has already been performed through one of the existing frameworks/approaches for WS discovery (e.g. [21]). Additionally, the user will have to setup few parameters to guide the filtering and ranking process (further details in Section 3.3.3).
After processing all the trustworthiness-related information associated to those WSs (by querying the sources), the ES will return a filtered and ranked list where each entry contains the following data:

- **Reference of the WS**: a unique reference to the service
- **Score**: the overall score given to the service

There will be cases where some WSs where not classified because of some missing information (e.g. the QoS offer does not exist). In this case the evidence supplier will put all those WSs in a list named **unclassified**.

### 3.3.2 Workflow

Figure 3.2 shows the workflow of the "evidence supply" step of this framework.

The flow starts by supplying a set of functionally matching WSs to the ES. From that point on, the user might potentially choose a WS even without any evidence of the WS trustworthiness (i.e. trusting it just based on the functional contract). Otherwise, the flow enters a sequence of steps where the user will have to set the guidelines to rank and filter the WSs, until the evidence supplied

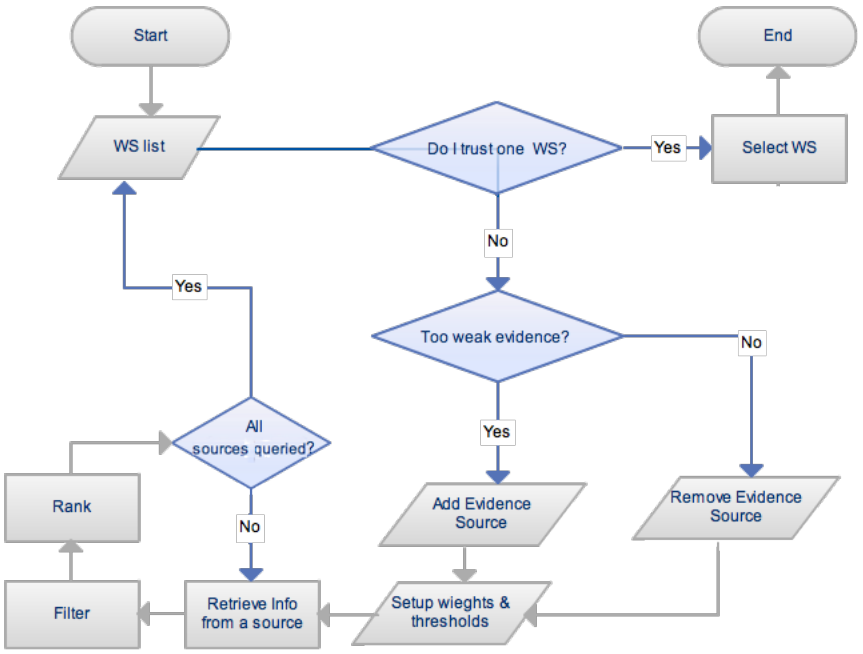


Figure 3.2: Evidence Supplier - Flowchart

is strong enough to make him take a choice regarding the WS to consume (or to negotiate a contract with).
The flowchart is supposed to provide just an intuitive idea of how the evidence supplier is involved in the trust-building procedure. Specific cases like those where the user is not able to make up his trust belief even after querying all the sources are not represented. They are not useful to understand the rationale, thus they will be discussed separately later in the document (Section 4.2.3).
The framework will actually involve two sources, but to stress the extensibility of the idea, the flowchart has been kept generic.

### 3.3.3 User-defined parameters

The ES has a filtering and ranking role. Given a set of services $\mathbf{S}$, it will return a ranked subset $s \in \mathbf{S}$ after retrieving trustworthiness clues from different sources. The user has then to be allowed to setup his own preferences to conduct the filtering and ranking process. There are two categories of parameters the user can set:

- **Weights**: the weights are associated to each specific evidence source. The user, depending on his attitude, might be willing to rely more on one source than the other ones. The *evidence supplier* has thus to provide a result satisfying this will.
  Weights are expressed as a percentage: in general, with N sources of trustworthiness, the user is allowed to set a weight on each of them so that the final sum of weights is 100%. In this case, sources of trust are QoS contract, reputation and direct experience. The user can even decide to “unplug” one of these sources. In such case the weight will be 0% . Formally:

  **Definition 3.6 (Source weights constraint)**
  Given $\mathbf{E}$ the set of sources and $w_e$ the weight given to evidence source $e \in \mathbf{E}$

  $$\sum_{e \in \mathbf{E}} w_e = 1$$

- **Thresholds**: the user is allowed to setup a threshold to meet on different sources. For instance, the user might desire a minimum reputation of 42%. Then the user can choose:
  - A threshold for each source queried by the evidence supplier.
  - A global threshold, that will represent the overall minimum score of the service after the weighed filtering and ranking have been applied by the *evidence supplier* to the service list.

  WSs that do not meet the threshold are automatically assumed to have the *minimum value* in the rank. All the threshold are expressed as a percentage.

  **Definition 3.7 (Threshold constraint)**
  Given $t_e$ the threshold assigned to the evidence source $e \in \mathbf{E}$:

  $$\forall_{e \in \mathbf{E}}\ 0 \leq t_e \leq 1, t_e \in \mathbb{Q}$$

Further parameters tightly related to the specific trustworthiness source can be set by the user, like the QoS demand. They will be described on the related sections.

### 3.3.4 Filtering and Ranking

The filtering and ranking process depends on the information supplied by the different sources. Moreover, the user, as described in 3.3.3, has previously set

specific thresholds and weights.
Given those premises, the filtering process is:

**Definition 3.8 (Filtering by threshold)**
Inputs are a *service score* and a *threshold*, output tells whether the service has to be filtered or not:

$$Filter?(sc, t) = \begin{cases} \text{true} & \text{if } sc \leq t \\ \text{false} & otherwise \end{cases}$$

For a given evidence source $e \in \mathbf{E}$ and the associated threshold $T_e$ , the set of Web Services filtered by means of $e$, $\mathbf{S}_e^f$, is:

$$\mathbf{S}_e^f = \{i \in S : Filter?(sc_i, T_e) = \text{false}\}$$

After the the service list has been filtered for each source, the weighted rank is performed, based on the scores given by the trustworthiness sources:

**Definition 3.9 (Weighted score)**
Given

- $W$ the set of weights for $\mathbf{E}$, where $w_e \in \mathbf{W}$ is the weight for evidence source $e \in E$
- $r_{e,s}$ the rating for service $s \in \mathbf{S}$ returned by evidence source $e \in \mathbf{E}$

The weighted score (rating) for a service $s \in \mathbf{S}$ is:

$$WeightedScore(s) = \sum_{e \in \mathbf{E}} w_e * r_{e,s}$$

Then, given $\mathbf{S}_e^f$ the filtered Web Service set for evidence source $e \in \mathbf{E}$, the ES take a list of services $\mathbf{S}^f$ such that,

$$\mathbf{S}^f = \bigcap_{e \in E} \mathbf{S}_e^f$$

where $\mathbf{S}_e^f$ is the list of Web Services filtered based on the result of evidence $e$. The ES then applies a sorting procedure returning the same list sorted by increasing WeightedScore.

In our framework there are currently three evidence sources for the evidence

supplier: the QoS, the Reputation and the Direct experience based ones. Anyway, the given definitions are generic enough to be applied to any set of sources where the returned result is represented by monotonically increasing scores list (better scores for higher values).
It is important to point out that the user should be able to do not consider a source at all by giving a 0 weight to it. In this case, the ES should not event query the source.

## 3.4 QoS based step

To date, many studies have analyzed the Quality of Service as a critical factor to discriminate different offers of the same WS capabilities. In [89] the authors claim that *QoS support for Web Services will play an important role for the success of this emerging technology* and *the QoS a service provider delivers will become a decisive criterion when services with the same functionalities are available at customers' choice.* Kyriakos, in *Mixed-Integer Programming for QoS-Based Web Service Matchmaking*[53], identifies the QoS augmented WS discovery as the main solution for filtering and selecting between functionally equivalent WSs. The authors of [88] disclose the results of a survey performed in UAE targeting IT managers of businesses in different sector. The statistic shows that QoS requirements are among the most important features when it comes to consume Web Services. In [63], Yan Lu et al. suggest an algorithm for dynamic composition of WS based on QoS.
We have mentioned just few of the whole list of studies regarding the QoS selection of Web Services. Therefore, Quality of Service should be considered as a source of trustworthiness: whenever a user needs to choose among plenty of functionally similar WSs, the knowledge of how the service will behave under a qualitative point of view may result in a critical factor for the final choice. For instance, a user might be willing to trust a service only if it can keep a huge amount of traffic. Thus, he will trust a service that assures a big *capacity*. In this section methodologies suggested by different papers and studies will be integrated in order to provide a selection and ranking mechanism for WSs based on their QoS contract.

### 3.4.1 Model

The model we are going to describe has been kept abstract enough to depend as few as possible from the technology it will be applied to. Thus, for instance, we are not going to list which attributes should be used or whether the specification

will be mapped in a XML or JSON document: this would imply assumptions on the underlying infrastructure. A low packet loss QoS requirement might, for instance, be a desirable attribute when working on a wireless environment, but not necessarily otherwise. Or even talking about a JSON schema would address the model to a REST-ful Web Service infrastructure.
The purpose of this chapter is, instead, to provide a generic framework the can be instantiated to (almost) any the required need.

### 3.4.2 Issues & Requirements

The growing literature related to QoS selection and ranking of WS demonstrates that this is not a trivial topic. There are many possible choices that can be considered when designing a QoS selection system for WSs. Each choice solves some specific problems, leaving others uncovered. The following list aims at categorizing the range of possible options for a QoS ranking system design, highlighting shortcoming and pros for each of them.

**Preferences vs Constraints:** in order to provide an automatic selection/ranking of the WS QoS offers, the potential WS consumer is required to specify his needs so to match them against the provider offers. There are essentially two ways of expressing those needs: weights for every QoS attributes and/or specific user-defined constraints. Works like [63] provide a ranking function where preferences submitted by the user represents the *demand*. This approach makes the ranking process more efficient. However it lacks of expressivity since the user cannot precisely define values specifically related to each attribute (e.g. $responsetime = 4.2ms$) and, moreover, all the QoS attributes must be represented as numbers. This implies that QoS features that are not directly described has numerical (e.g. security) must be "translated" by means of a specific function (e.g. security $\rightarrow$ level of security [63]).
Other studies try to address the problem of ranking/filtering by matching a constraint-based demand against the offers [53][88][91]. This gives to the client a better expressivity in defining the non-functional requirements. There are anyhow some issues to consider:

- *the more fine-grained (expressive) the contract is, the lower the efficiency* of the matching process results. [53] is dedicated to issue of efficiently compute the QoS matching process with symmetric model for offers and demand and non-linear constraint support;
- *the lower expressive the contract is, the lower the precision*: [88] describes a model where offers and demand are expressed as vectors

of real numbers. Unlike the previous example, with this model is not possible to define a range for a given attribute (e.g. 1.0ms <= *response time* <= 34.0ms). The algorithm is then extremely less complex and probably more efficient. However such kind of approach does not allow to discriminate with super-optimal solutions and sub-optimal ones. The result just depends on the similarity with the offer, no matter if an offer is dissimilar because better or because worse.

Other works apply an hybrid solution, where a previous selection based on a constrained demand approach is enriched with a further ranking based on preferences [97].

**Ranking vs Filtering:** Approaches like the one explained in [79] are supposed to provide a filtering criteria for a given set of WS QoS offers. That is, depending on the demand, the function classifies the offers in two groups: matching and not-matching ones. Others, instead, aim at sorting the offers depending on a preference or similarity function ([63]). Usually the integration of the two procedures is adopted: a filtering step is applied to keep a list of just the matching services. Essentially a sorting step is applied, to rank the remaining offers depending on a user specified dominant attribute ([97] [91]). Other approaches provide a richer filtering procedure where offers are classified in few categories. Like, for instance, as non-matching, exact-matching, super-matching ([52]). Shortcomings and pros of the two approaches usually depend on other choices. For example, if a constrained demand is used, it might result difficult to apply a ranking method: in which cases an offer is better then another one? If an attribute demand is specified like $q_i > 42$, how two offers with $q_i^1 = 43$ and $q_i^2 = 50$ should be interpreted? Equally good, $q_i^1$ better or $q_i^1$ worse than $q_i^2$? This might depend on $q_i$ and on the user intentions. Indeed, in order to have a good constrained approach with ranking, a strong semantic for the QoS specification should be defined.

**Single service vs Composition:** The QoS discovery and selection procedure may be performed with different goals. The most part of the studies considers each WS selection as an independent event. This is indeed a common use-case when it comes to find the best service to consume on the fly.
In different use-cases, a user might be both a consumer and a provider himself and thus aiming at creating a WS composition in order to expose another service (or just for the internal enterprise use). In this case, the selection function executed should take into account not just the QoS of a single WS, but of the entire composition, even accepting a less then optimal choice for one WS in order to achieve a global optimum for the choreography. Notable works in this direction are [104] and [63].

**Fixed vs Flexible attributes:** Another important distinction between different studies is outlined by the range of considered QoS attributes. Usually the suggested frameworks include specific attributes just for testing or example purposes, but they adopt a model or a specification that can be extended on demand (like [97] and [89], where the proposed solution relies on the well known *tModel* schema[2]). Krikitos et al. in [51] provide an ontology and a alignment algorithm to handle the semantic matching between QoS attributes. [104], instead, provides an optimization model built around the existence of four precise QoS attributes. The drawback of this kind of approach is that it is hardly extensible to different subdomain, but on the other hand it results fully optimized for a specific case.

Out of this analysis it is possible to see how one perfect universal choice is difficult to take. It would be out of the scope of this document to figure out the best solution for all the cases. The aim in this project is just to provide a basic QoS preliminary selection capable of helping the user understand what to trust. It is not therefore required to provide the finest-grained QoS specification or the best ranking function: that would be a standing-alone research topic. Although there are some requirements that should be met in order to make the QoS based selection component fit the framework:

- **Generalization of the attributes**: as we stated at the beginning of this chapter, the framework should be generic enough to be applied to the widest possible range of SOA contexts. Thus the QoS model should be kept abstract from the specific attributes and from the specification language. In other words, it is not possible to tie the matching algorithm to some QoS predefined attributes and semantic.
- **Ranking**: in order to collaborate with the *Evidence supplier*(3.3), the QoS based step should return a list of WS references ordered by offer fitness to demand. Then, each WS reference returned should be scored with a numerical value that will be integrated to the results of other trustworthiness sources. Moreover, it is not required to provide a filtering technique: recalling the definition given for *trust* (Def. 3.4), it is necessary to leave the user the final choice, even if not rational (so he might be willing to select the poorest matching offer just to help the provider in some kind of beta-testing).
- **Complexity/precision tradeoff**: as stated during this analysis, the aim of this project is not the study of the QoS selection at a low level. The QoS selection is just a preliminary step towards a more complete solution. The user should be provided with a suggestion about the choice to take, but

[2]`http://uddi.xml.org/tmodels`

this QoS evidence will be enhanced by other components in the system. Thus, even to meet the time limits and the integrability requirements, the adopted QoS selection framework has to provide a preliminary ranking of a list of WSs without falling in a complex solution that would result in a stand-alone study.

Based on these premises, the next section will explain the decided solution for implementing the QoS based ranking component.

### 3.4.3 The QoS Ranking

As we showed in the previous section, many studies in literature addressed the problem of QoS matching, ranking and selection for WSs. They differ in many aspects: adopted algorithms, level of abstraction, expressivity of the model, languages and standards and so forth. Our attention was addressed to those studies that could have been easily integrated in our framework.
In [91] a complete architecture for the QoS based WS selection is described. It is a notable work, published in the *International Journal of Computer Applications* in 2010, that does not meet the requirements for this component of our framework though. The main weakness is the lack of abstraction: the study has been addressed to SOAP WS. For instance, the existence of the UDDI registry is assumed. Then the *tModel*[3] is used as the data structure to represent QoS offer/demand. Moreover, the matching and sorting algorithms are left uncovered and thus it was not possible to extrapolate them for the reuse in our framework. Another work related to QoS based WS selection is *A Framework and QoS Matchmaking Algorithm for Dynamic Web Services Selection*[88], proposed by L.Taher et al. in at the *Second International Conference on Innovations in Information Technology*[4] in 2005. A plus of this work is the separation of the framework in two models: the data model and the computation model. The former include all the data structure used to handle the QoS properties and semantic, like ontology registries. The latter concerns the matching algorithm and the management of the QoS dynamic changes. As stated by the authors, this separation has been carried out in order *to provide a generic framework that can be customized for any domain.* Therefore, even if a consistent part of the framework is instantiated to SOAP WSs, with all the schema specifications, the algorithmic part is reusable for any domain. WS QoS offer and demand are represented by vector of real numbers, where each dimension corresponds to a QoS attributes. After a normalization step, the Euclidean distance between

[3]`http://uddi.xml.org/tmodels`
[4]`http://www.it-innovations.ae/iit005/index.php`

each offer and demand is applied in order to find the most similar (and so, the most fitting) offers. This procedure, however, exposes various issues:

- QoS offers and demand are expressed as points in the space. That is, there is no way for the consumer or the provider to express a range of values for their QoS preferences. This could be indeed useful, in particular when it comes to deal with QoS measurements: response time is not usually a constant value, so both the provider and the user might be willing the have a tolerance interval for such kind of metrics. Moreover, it would be rather difficult for certain users to establish an exact value for QoS parameters. Should a value higher than the one specified be considered as better or worse? Questions like this one are left unanswered in the study.
- The Euclidean distance has some properties that turn advantageous for what concern the user QoS demand specification. The computed distance will reflect the similarity between that query and the offers regardless of the dimensions of the vectors. This way the user is not constrained to specify a preference for each QoS attribute, still receiving a valuable answer. Nevertheless, this exposes the problem in case of poor QoS offer specification: if the QoS offer vector has a lower dimensionality than the demand one, a good match can be achieved anyway. So, if the provider creates a 1 attribute QoS offer and that attribute value meets the user demand, there will be a good match, even if the user had specified many other different parameters.

A partial solution to the problems stated above would be to change the QoS demand model: instead of defining precise values, that might be a non-trivial task, the user should be asked to input preferences for each attribute. Practically speaking, it is already known which values are better for a given parameter. For instance, the *response time* is a dimension that improves by decreasing its value. Thus, the lower the value is, the better the performance results. In most cases the user might be just not interested in a certain QoS attribute. In that case, he should apply a low weight. But whenever the potential consumer wants a QoS parameter to be considered, it is a good approximation to assume that he will prefer generally high values for monotonically increasing dimensions and lower values for monotonically decreasing dimensions.
This concept is partly applied in various studies: in [97] the authors rank (after a filtering step) the QoS offers depending on the value of the user-selected dominant attribute. Selecting a dominant attribute is the same as giving the maximum weight to it whilst assigning weight 0 to the remaining ones. This makes sense in this work because there is a previous filtering of the QoS offers based on some user specified parameters (that let the user be more expressive). A more suitable solution for the user preferences specification is proposed in

[104]. Even though the paper addresses the problem of multi-granularity WS composition (out of the scope of our project), there are few parts worth to consider for the integration in our system. Before explaining how, the relevant features of the framework will be briefly introduced:

- A limited set of predefined QoS attributes is used;
- A process is mapped to multiple WSs interacting with different patterns;
- Service classes are considered to group different WSs with similar functionalities;
- An optimization model is suggested to solve a QoS-based Multi-Granularity Service Selection Problem;

The part of the paper considered for our project is the one describing the optimization model. The authors adopt an objective function where each variable represents a QoS attribute. The number of variables is fixed. Then they define few constraints to help the discovery of the best composition. An adapted version of the objective function will suffice in our framework.
First of all, the constraint used by the authors of [104], are specifically conceived for the QoS attributes they decided to consider. Thus, to meet the *abstraction* requirement advocated throughout section 3.4.2, they are not going to be included in our QoS ranking component because they would tie the algorithm to the QoS details. The remaining part is a maximization function returning the best scoring QoS offer calculated on the weighted sum of the normalized scores of the different QoS attributes:

$$max \sum_{i=1}^{4} w_i \, * \, Norm(Q_i)$$

This is not fitting to our framework though: it is limited to four QoS attributes. Therefore, the first modification is the substitution of the constant limit to a limit equal to the amount of attributes $q_n$ specified in the demand.
Then, the final goal of [104] is to find the best *path* to achieve the a result with the best QoS. Therefore, an optimization problem has to be solved. In our study, a sorting problem has to be addressed. So, while keeping the same fundamental idea, the original algorithm has to be revised. The need of our QoS component is to obtain a score for each QoS offer, depending on the user defined weights. The score function is then defined as follow:
Given

- $Q$ the set of all the possible QoS attributes;

- $I$ a set of indexes, where $|I| \leq |Q|$ and $\forall_{i \in I}\ q_i \in Q$
- $W$ a set of user-defined weights, where $|W| \leq |Q|$ and $W \in \mathbb{R}^{|W|}$;
- assumed $w_j \in W$ the $j_{th}$ weight in $W$, $i_j \in I_D$ is the $i_{th}$ index in the demand index set $I_D$, then $w_j$ is the user-defined weight for $q_i \in Q$;
- a QoS offer $O$, where $|O| \leq |Q|$ and $O \in \mathbb{R}^{|O|}$;
- assumed $o_j \in O$ the $j_{th}$ offer value in $O$, $i_j \in I_O$ the $i_{th}$ index in the offer index set $I_O$, then $o_j$ is the provider-defined offer value for $q_i \in Q$;

$$\sum_{w \in |W|} w = 1,$$

**Definition 3.10 (QoS score function)** For an offer $O$ and weight demand $W$,

$$Score(O, W) = \sum_{j \in |W|} (\ w_j * Norm(O, i_j)\ ), w_j \in W \ and \ i_j \in I_D$$

The function $Norm(O_q, q_i)$ is a normalization function. It is inspired by the one suggested in [104] and [97], where the authors distinguish decreasing dimensions and increasing dimensions:

**Definition 3.11 (Increasing dimension)** The quality of the dimension improves with the increase of the value.

**Definition 3.12 (Decreasing dimension)** The quality of the dimension worsens with the increase of the value.

The function has been slightly modified in order to support the case where a required attribute is not present in the offer. In that case it is necessary to outdistance that offer from those having a value defined. The goal is achieved by shifting the score of all the offers *having* the attribute. This way they will all have a value proportional to the given weight, whilst leaving to 0 the other missing the attribute (no weight in the total score). The shift for the moment is a variable $\gamma$ whose value should be established depending on how much missing attributes has to be outdistanced by the present ones. This would need some practical test to be decided.
It is even necessary to define a function to map a QoS offer attribute $o_i \in O_q$ to the corresponding attribute $q_i \in Q$:

**Definition 3.13 (QoS Mapping function)**

$$FindOffer(O,q) = \begin{cases} o_i \in O & \text{if } \exists i \in I_O\ t.c.\ q = q_i \\ null & otherwise \end{cases}$$

Given

- $max_i$ the maximum value for an attribute $q_i \in Q$ among all the offers for that attribute;
- $min_i$ the minimum value for an attribute $q_i \in Q$ among all the offers for that attribute;

The normalization function is then:

**Definition 3.14 (QoS attribute normalization)**

$$Norm(O,j) = \begin{cases} \frac{FindOffer(O,q_j)-min_j}{max_j-min_j} + \gamma & \begin{array}{l} FindOffer(O,q_j) \neq null \wedge \\ \text{if } q_j \text{ is increasing } \wedge \\ max_j - min_j \neq 0 \end{array} \\ \frac{max_j-FindOffer(O,q_j)}{max_j-min_j} + \gamma & \begin{array}{l} FindOffer(O,q_j) \neq null \wedge \\ \text{if } q_j \text{ is decreasing } \wedge \\ max_j - min_j \neq 0 \end{array} \\ 0 & otherwise \end{cases}$$

Finally, for a list o QoS offers $\mathbf{O} = \{O_1 \ldots O_n\}$ and a set of user defined weights $W$, it is trivial to define the ranking function:

**Definition 3.15 (QoS ranking function)**

$$Rank(\mathbf{O}, W) = \{\mathbf{O}_s : \forall_{i \in |\mathfrak{O}_s|-1} Score(O_i, W)\ \leq\ Score(O_{i+1}, W)\}$$

Where $\mathbf{O}_s$ is the original offer list sorted by *Score*.
In case a service provider did not arranged an offer for his WSs, they will be labeled as *unclassified*, so to make clear that they do not take part to the rank.

## 3.5 Reputation based selection step

The second step of the WSs filtering and ranking involves a reputation framework. This will constitute the *social* enhancement of the trustworthiness evidence provisioning. Or, to use the words of the second chapter, the social part will help the user build the *soft trust* towards the different services. As already analyzed in Section 2.3.2, many studies have suggested reputation systems for trustworthiness provisioning. Some of them suggest centralized approaches [65], but, as we stated in section 2.3.5, a centralized system does not suite a SOA environment. Other works propose distributed reputation systems, like EigenTrust[48]. The approach does not directly address the WS environment, but it is studied in a P2P file sharing context. In [39] QoS reports are used as the feedback for reputation framework based on a P2P network. It focuses on the cheaters recognition, suggesting methodologies to identify dishonest reports and dishonest users. The framework, though, does not fit ours because reputation is tightly connected to a QoS report of a WS, whilst our reputation rates are going to be computed based on different parameters.
Other studies define hybrid systems where reputation is used to mitigate the issues of other approaches. In [98], the author suggests a model where a preliminary WS QoS discovery is enhanced by a reputation system, but the reputation scores are assumed to be trustworthy.
A notable work addressing the reputation management issue for WS is [61]. It explains a distributed architecture based on the *referral* approach(Section 2.3.2.1). The architecture and rationale are illustrated in the next section.

### 3.5.1 Referral Based Reputation Framework

As mentioned in the *Previous Work* section of *A Distributed Reputation Broker Framework For Web Service Application*[61] , the framework has been influenced by studies related to *referral network systems.* The actors on behalf of the referral in this system are the so-called *Trust Broker* (TB). The other entities are the *users* (that actually consumes the services) and the *reputation authorities*, who are the “final chance” to obtain a reputation rating for a service (more details later on). Figure 3.3, shows how these entities collaborate.

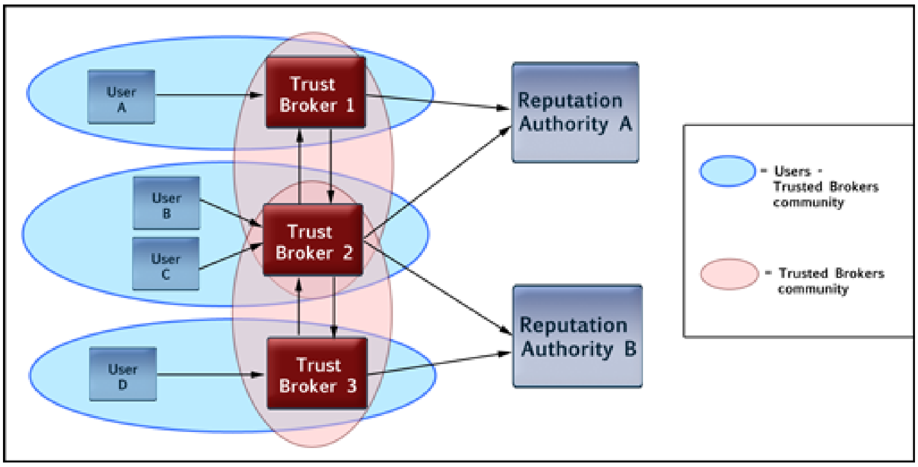


Figure 3.3: Entities relationships

Briefly, the workflow of the system is:

1. a user *A* needs to retrieve the reputation of a user/service *B*;
2. *A* contacts the desired *TB*, specifying the following informations in the request:
    - requester ID (*A* in this case)
    - service ID (*B* in this case)
    - transaction count threshold (the number of transactions the reputation score has to have been calculated on)
3. the *TB* verifies whether it owns a reputation score for the service computed on the basis of enough transactions (the threshold specified by the requester);
4. if it does not, it contacts other trusted *TB* to obtain the required information;
5. if they do not manage to provide this information either, it contacts a *Reputation Authority*, that it is supposed to collect informations about all services;
6. finally, when the reputation rating is retrieved, the result is returned to *A*;

7. *A* can now choose whether to perform the transaction or not; in case it does, it can return a reputation score back to the trust broker, who will update its database.

The *TB* is designed as in figure 3.4: The two main components are the *Reputation Manager* and the *Connection Manager*, described below.

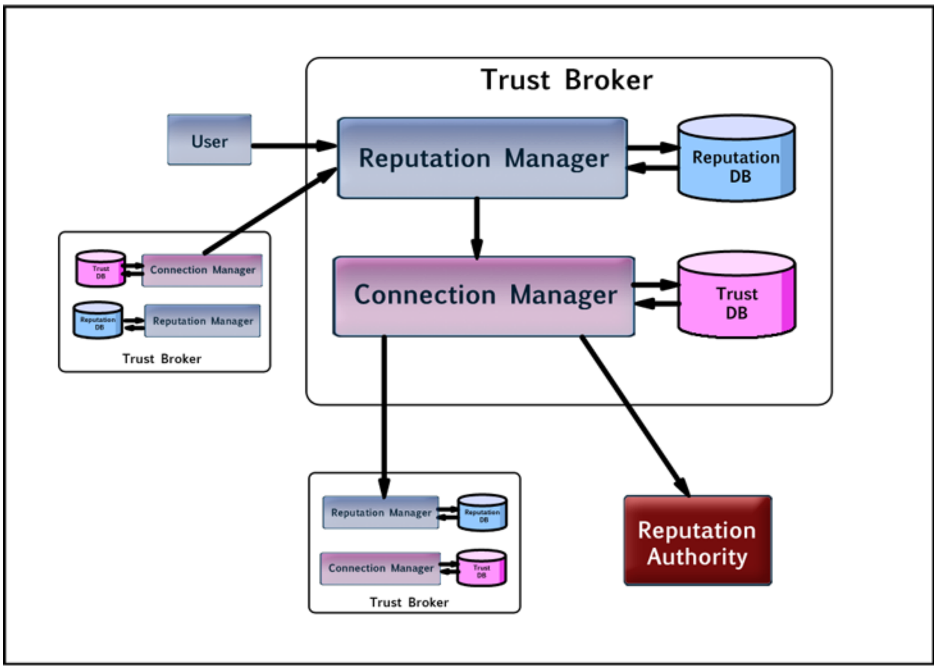


Figure 3.4: Trust Broker

### 3.5.1.1 Reputation Manager

The reputation manager has different functionalities. It acts as the interface for the *user*-*TB* communication. This means that, whenever a user needs to retrieve a reputation score or to submit a rating, he will contact the *reputation manager* of the *TB*. As a consequence, the reputation manager is in charge of keeping and aggregating the users' ratings about their experiences and providing scores to the requesting users. Two formulas are suggested to calculate the reputation update on a new rate submission.

**Definition 3.16 (Time dependent reputation update function)** This function assumes highest weight for more recent feedbacks. This allows users to have a better expectation on the current service performance.

$$R_{new} = e^{\beta \Delta t} \frac{N}{N+1} R_{old} + (1 - e^{-\beta \Delta t} \frac{N}{N+1}) r$$

Where

- $r$ the new feedback
- $R_{old}$ is the latest computed rating
- $\Delta t$ is the time between $r$ and $R$
- $e^{\beta \Delta t}$ the discount factor for $R_{old}$
- $N$ is the number of already computed updates

The discount factor will determine the window out of which a transaction rating is not meaningful any longer. During the performance study, for instance, the author set a window of 100 transaction by fixing $\beta = \frac{1}{36*10^5}$.
By providing a $\beta = 0$, the time will not influence outcome of the function:

**Definition 3.17 (Time independent reputation update function)** This function assumes the service quality is not affected by the time, so $\beta = 0$

$$R_{new} = \frac{N}{N+1} R_{old} + (\frac{1}{N+1}) r$$

When a not sufficient evidence of a service reputation is owned (this means the requesting user threshold is too high), the *reputation manager* delegates the *connection manager* to contact other brokers.

#### 3.5.1.2 Connection manager

The connection manager is the entity in charge of dealing with the network of trusted brokers. It keeps a list of trusted brokers and updates the trust relationships with them. It collaborates with whom it trusts more to obtain reputation scores (when the *reputation manager* requests them) and it provides reputation informations to other brokers. Trust towards a broker depends on the number of accurate recommendations that the broker has provided. It is a value in the range $[0, 1]$. 0.5 is assumed to be a neutral value (starting value).

**Definition 3.18 (TB Trust increment)**

$$X = X + F * (1 - X)$$

**Definition 3.19 (TB Trust decrement)**

$$X = X * (1 - F)$$

F is a positive index less then 1. The functions are designed to make the trust gain more difficult then the trust loss. Its value is not fixed even in the performance study of the paper: a random value within the range [0.2, 0.5] is picked-up for each TB in the test phase. This is because tests have been more focused on the evolution of the system correctness depending on initial trust values among TBs and initial local reputation scores for the tested services. Thus the F has not a critical role in the system overall correctness and can be arbitrarily chosen among a range of intermediate values (like the authors suggest).

#### 3.5.1.3 Discussion

Many details are indeed missing, but it would be out of the scope of this document to precisely describe the whole solution. For further informations we strongly suggest to read the original paper [61].
It is instead more important to focus on the worth points of this work, that led us to the choice of integrating it in the framework.

- **Light Client**: considering an environment where the service consumer might be hosted in a mobile device, considering the hardware limitations and the battery consumption, it would be desirable to keep as much of the computation as possible on a different server. Thus, the distribution of the logic on dedicated *Trust Brokers* meets this requirement.

- **Easier rating collection**: in a pure P2P network, where each user interacts with other users, it would be more difficult to collect enough ratings to achieve a reliable reputation value for the services. Thus, collecting more ratings in a *Trust Broker*, helps reaching a worth result with less effort;

- **Faster computation**: again, in a pure P2P network, requesting reputation scores to each user would increase the number of connections (and then the time) needed before obtaining a valuable score. With trust brokers constantly collecting ratings, this issue is weakened. Moreover, part of the scores can even be computed offline.

- **Time dependent function**: one the reputation update functions takes into account the timestamp of each rating. This methodology enclose an interesting consequence: reputation is fresh. If a service had a good reputation for a long period and than, all of a sudden the service encounters some serious problems and stop providing the service as supposed to, the reputation score is affected and all the potential consumers can figure out something is going wrong.

- **Anti-collusion**: if a malicious broker joins the community, at a certain point it will probably start providing recommendations that do not match the honest ones. The other brokers will then recognize it and progressively stop collaborating with it.

- **Performance study**: the framework has been actually implemented and tested with different parameters to verify it correctness. Results show that a system initially setup with neutral or good reputation scores in TBs reaches the overall correctness over the time, whilst a system configured with bad initial reputation scores tends to remain not correct (no one tries the WS because assumed to be untrustworthy and then reputation scores are never updated). Thus, the system behaviour is known, even if we are not going to provide an implemented solution in our project.

Even if there are many advantages in this solutions, some issues are not addressed, like.

- A user is assumed to be honest. So he can possibly poison the reputation database of a broker. No mechanism to recognize wrong user rating are provided.

- As analyzed in chapter 2, a WS is not alway a one-time-use-resource. In many occasions it happens that a WS is part of a more complex system exposed as another service. It is then going to be used quite frequently in a short time interval. In the paper the authors assume the user requests a reputation score and then responds back with the feedback about the service invocation. In case this procedure has to be performed a lot of times, there would be a great overhead and few efficiency.

### 3.5.2 Integration with the Framework

The integration of the reputation system with our framework has to be accomplished considering 4 points of concern, related to algorithmic choices, components interaction and so forth.

#### 3.5.2.1 Reputation update function

Which of the two reputation update functions should be adopted to best suite our framework? Considering the W3C use-cases, we believe that there can be users for both of them. Function 3.16 is more likely to be useful for direct consumers, i.e. users that will select the WS and consume it directly. In this case, if the service is experiencing temporary problems, it would turn desirable for the user to know about it. On the other hand, if a developer-user is composing a service-oriented infrastructure, the service is going to be used repeatedly and for a long time. Thus, it would be probably more useful to know the overall reputation of the WS, regardless of a temporary failure. Therefore, both of the functions should be available and the consumer should evaluate and select which is the most fitting to his needs.

#### 3.5.2.2 Reputation request

The *trust-broker* computes the current reputation score for each WS. It receives a parameter in the form:

$$< \; user\_id, \; ws\_ref, \; min\_transactions >$$

To decrease the overhead and improve performances, the input parameters are extended so to provide a list of WS references instead of a single one. Furthermore, in light of the discussion in 3.5.2.1, the user should specify whether he prefers a time dependent or independent score.

$$< \; user\_id, [ws\_ref_1, \ldots, ws\_ref_n], min\_transactions, is\_time\_dependent >$$

Of course the *trust broker* has to be instructed to compute the reputation score for each of the WSs, to then return the two list as explained above.

#### 3.5.2.3 Reputation-agent / Evidence-supplier interaction

The reputation system and *evidence supplier* have to communicate through an agreed interface. As mentioned in 3.3, the *evidence supplier* is going to deal with tuples containing the following data:

- Ranked list of WSs whose score has been successfully calculated
- List of web services whose score has not been calculated and are then **unclassified**

Considering the *trust broker* is computing a reputation numerical value for each requested WS, the rank is trivial: higher the reputation score, higher the ranking. The WS without enough transactions are going to populate the *unclassified* list.

#### 3.5.2.4 User ratings

The last integration point to think about concerns the consumer-generated ratings: how can a service be rated? Considering that the framework should be adaptable to the widest range of possible use cases, the following scenarios has to be taken into account:

- The system can be used by a sporadic consumer, that means the service is selected, directly used by the consumer and stop. In this case the user should submit a score based on his experience (even manually).
- The system can be used by a developer to compose a service oriented system based on a choreography. This means each service is going to be indirectly used more times by the users consuming the main system. In this case there are two points to focus:
  - the rate cannot be submitted by the end user because he is transparently using the main application unaware of the underlying infrastructure. Therefore the rate has to be **automatically** generated and submitted by means of an agent. Thus, the agent has to be driven in the rating process by some rules, disguised as a contract.
  - a WS participating to a composition is consumed more frequently in a time interval by the same user (the one identified by the developer) than a service directly invoked by a sporadic consumer. This leads to

> a conclusion: it is more likely that the service behaviour will not be changing drastically and continuously in a shorter interval. Therefore, it would be a waste of resources to provide a feedback every time the service is consumed. The idea is then to send one feedback each predefined time interval or in case of considerable changes in the local rating.

All these considerations boil down to this idea: the rate can be submitted both manually by the user and by an agent that automatically checks the conformity of the WS interaction to the contract. This second case will be better discussed in 3.6.
The *trust broker* needs, anyway, a rate and a timestamp. In light of what has been outlined, a rating should be expressed following the definition 3.20.

**Definition 3.20 (Service rating)**
A service rating is an evaluation of the web service conformance to the expectations. It is expressed as a tuple:

$$< \ user\_id, \ ws\_ref, \ rating, \ timestamp \ >$$

*rating* is a real number within the range $[0, 1]$.

## 3.6 Agreement negotiation and monitoring

As a final step towards building a *trustworthiness* belief, we advocate that a contract negotiation is necessary. This will allow the consumer to formally establish an agreement directly with the service provider. By means of an agreement, the service consumer can systematically, constantly and automatically evaluate the outcome of his interactions with the service. This evaluation can then influence the user own trust towards the service (weakening or enhancing it) and also be used as a score to provide to other users willing to consume the same service. In our framework this score may, for instance, be submitted to the reputation manager or be used as a *direct* evidence of trustworthiness.
In this section we will explain how to agreement is established and how it will influence the future experience of the user.

### 3.6.1 Motivation

As analyzed throughout Chapter 2, *automated trust negotiation* is one of the main families among the ones existing in literature dealing with the WS trust provisioning issue. It solves some of the problems arisen during the analysis:

- it always produces a tangible result, depending on the negotiation outcome: if it fails, the security parameters offered by the provider do not satisfy the user needs (or, the other way around, the consumer demand does not fit the provider offer). Thus, not trust relationship is established. Otherwise, once an agreement is achieved, the consumer and the provider are trusting each other.
- both provider and consumer disclose their security behaviour iteratively and incrementally. This improves the privacy safeguard for both the provider and the consumer: sensitive informations are kept locally. They will be exposed step by step with the growth of the trust between the two peers.

Another advantage of this approach is that the trust relationship is also established from provider towards the consumer. Anyhow, this topic is not going to be analyzed in this work, because what we want to focus in this project is the trust establishment from the consumer to the provider.
In Sections 3.5 and 3.4 two techniques to rank WSs have been illustrated. The first one is based on a social *soft trust enforcing* approach, the second one relies on a *hard notion* of trust. However, we believe the *hard trustworthiness* component in the framework does not suffice. Trust negotiation is necessary because a QoS contract is not rich enough to express all the feature related to the security behaviour of a web service. Moreover, it does not provide any way to establish the access control rules for the service use and a formal way to monitor the service behaviour. Let us provide two use cases to make the point:

- Alice is willing to use a flight booking service that does not require the ID card disclosure. Alice picks up a WS selecting it on the base of its reputation and QoS. She perform the whole purchase procedure discovering, just at the end of it, that she has to provide the ID card number to complete the process. This kind of situation is indeed better to be avoided.
- A company would like to publish a WS for a call center management. The company does not want that everyone can indiscriminately know the service usage conditions. It will disclose them just to whom is going to disclose, in turn, part of his policies (a fair exchange of informations).

The use cases we illustrated require the two peers to negotiate a contract.
So far, most of the trust negotiation approaches applied to a WS environment are still an academic research field, where few works have addressed the issue

directly to a SOA context. The literature about that is growing, but still missing some important points specifically targeting a SOA.
A valuable work about Web Services and Automated Trust Negotiation is exhibited in [25] and further elaborated in [26]. The authors describe a framework for the negotiation of credentials but also the corresponding negotiation of services and the behavioral constraints on the disclosures of credentials depending on the business process.
The integration of this idea to our framework can fulfill the requirement of a contract negotiation.

### 3.6.2 Security by contract

Two main features distinguish [25] and [26] from other related works:

- **Real-world case negotiation**: the study takes into account the possibility of negotiating the disclosure of security privileges for more (additional) services[5];
- **Robustness**: fault tolerance techniques have been devised in order to make the framework fitting to an open environment such like SOA. This way both cooperative and malicious peers have been taken into account.

The papers then thoroughly explain how the negotiation is carried out once both consumer and provider had defined respectively the demand and the offer. In a nutshell, the service provider's offer looks like this: *I will grant you services $s_1 ... s_n$, but in change I want you to show me that you have security attributes (or privileges) $p_1 ... p_n$.* Further, *I will ask you to show me your credentials according to the following dynamic security rule $x_1$* where e.g., possession of privilege $p_i$ is asked before services can be accessed.
On the other side a client is making a counter-proposal: *I want to use your services $s_{0,1} ... s_{0,n}$ , and I am only willing to give you evidence that I have security privileges $p_{0,1} ... p_{0,m}$.* Further, *I am going to accept showing my credentials only according to the following dynamic security behavior $x_{0,1}$* where e.g., I am willing to show possession of privilege $p_i$ but only if I am asking service $s_1$ or $s_2$.
To give an idea of the rationale of this framework, the main functionalities and definitions will be illustrated. For further details we suggest to directly refer to the two papers.

[5]instead of allowing the negotiation of one service/resource only

#### 3.6.2.1 Services and Privileges

The basic components of a generic contract are *products* offered and the *goods* required to serve out the exchange. In SOC, products are the *services* and goods are the *privileges* or *credentials*. For instance:

- **services**: *getProduct*, *payProduct*;
- **privileges**: *passportID*, *creditCardNumber*;

In a nutshell, *services* are what a provider offers and *privileges* are what the consumer should provide in order to consume certain services. For the sake of abstraction, we consider both services and privileges as atomic predicates, without further details on specification and description languages.

**Definition 3.21** Services and Privileges Let $\mathbb{P}$ be a set of atomic propositions $p$ denoting security privileges and let $\mathbb{S}$ be a set of atomic propositions $s$ (disjoint from $\mathbb{P}$) denoting services.[25]

#### 3.6.2.2 Contract rules

Services and privileges are bounded by specific rules. For instance, provider and consumer could connect services and privileges like:

- **Consumer**: *I'm willing to disclose my 'department affiliation ID' only for service 'publish results' or 'run mobile code'*
- **Provider**: *Service 'publish results' is worth providing 'department affiliation ID' or 'research project affiliation ID'*

These are defined as *security behaviours* in [26]. Essentially, a security behaviour is a rule that specify which privileges are intended to be disclosed for which services. More formally, the grammar is:

**Definition 3.22 (Contract rule)**
$C_{rule} := \mathrm{S} \leftarrow \emptyset \mid \mathrm{S} \leftarrow P$
$S := S$ **and** $S \mid S_r$
$S_r := s_1 \mid s_2 \mid \ldots \mid s_n$
$P := P$ **and** $P \mid P_r$
$P_r := p_1 \mid p_2 \mid \ldots \mid p_n$

Example: $C_{rule1}$ = *payProduct* ← *email* and *creditCardNumber*.
To model the conversational nature of the interaction, a contract rule list is defined.

**Definition 3.23 (Contract rule list)**
A *contract rule list* is a non-empty list of contract rules related to different services:

$$\mathbb{C}_{rules} = < C_{rule1}, \ldots, C_{ruleN} >$$

Example:

$$\begin{aligned}\mathbb{C}_{rules1} = < & \, selectItem \leftarrow \emptyset, \\ & addToCart \leftarrow creditCardNumber, \\ & confirmTransaction \leftarrow email \textbf{ and } address >\end{aligned}$$

#### 3.6.2.3 Security policy

A set of contract rules along with the services and privileges appearing in the contract rules list, represents a *proposal.* A proposal is used within the negotiation to let the peers exchange their preferences regarding offers and demands.

**Definition 3.24 (Proposal)** A proposal *Prop* for a peer **A** is a set

$$< \mathbb{S}^A, \mathbb{P}^A, \mathbb{C}^A_{rules} >$$

where $\mathbb{C}^A_{rules}$ specifies the contract rules for the given set of services $\mathbb{S}^{\mathbb{A}}$ and privileges $\mathbb{P}^{\mathbb{A}}$.

Both provider and consumer can setup more then one proposal, so to manage more possibilities and preferences. During the negotiation, if a proposal is not acceptable for the other peer, another one is suggested.
The list of proposals identifies the *security policy SP* , i.e. the acceptable proposals for a peer. The preferences are expressed by means of a partial order over the security policy list.

#### 3.6.2.4 Agreement negotiation

In a nutshell, the negotiation between the consumer and the provider is an exchange of proposals until an agreement is met. The consumer kicks-off the conversation by asking for a set of services, and the provider replies with a proposal, specifying the required privileges. The conversation ends either with an accepted proposals (an agreement) or with a failure, in case there are no acceptable proposals for the two peers.
Whenever a proposal is sent by one of the peers, the other peer verifies whether it is acceptable by comparing it to its security policy set.
Let us define the consumer proposal as the demand $< S^D, P^D, \mathbb{C}^D_{rules} >$ and the provider proposal as the offer $< S^O, P^O, \mathbb{C}^O_{rules} >$. An agreement $\mathfrak{A}$ is a demand proposal $\mathfrak{P}^D$ that matches an offer proposal $\mathfrak{P}^O$. Formally:

**Definition 3.25 (Agreement)**

$$\mathfrak{A} = \{\mathfrak{P}^D \in SP^D \ : \ \exists\, \mathfrak{P}^O \in SP^O \ t.c. \ S^D \in S^O \wedge P^O \in P^D \wedge \mathbb{C}^D_{rules} \sqsubseteq \mathbb{C}^O_{rules}\}$$

Where $\sqsubseteq$ is a *match operator* such that $X \sqsubseteq Y$ returns true if behaviour specified by $X$ is among the behaviours allowed by $Y$, false otherwise.

## 3.6.3 Integration with the Framework

#### 3.6.3.1 Negotiator

The *Negotiator* component in our framework will be in charge of taking a user-defined set of proposals and use them to negotiate a contract for a given WS. The negotiation will be performed as described in 3.6.2. For the sake of simplicity, from now on the negotiation process will be defined as:

**Definition 3.26 (Negotiation)**

$$Negotiate(ws\_ref, proposals) = \begin{cases} \mathfrak{A} & \text{if an agreement is met} \\ \emptyset & \text{otherwise} \end{cases}$$

If an agreement is met, that agreement is returned and will be used as the contract.

#### 3.6.3.2 Monitoring

The contract obtained through the negotiation is used to automatically monitor the interaction with the WS. The original paper does not explain how the rules listed in the contract should be enforced during the ensuing conversation. Intuitively, monitoring the behaviour of a WS according to an agreement A requires to check that the actual conversation with the service is compliant with the claimed rules in A. From an architectural point of view a monitor can be realized by means of a specific component running at the client side that intercepts each incoming/outgoing message and checks whether or not the message is compliant with the agreement. In these terms, an agreement violation is like a protocol violation. The monitor notifies a violation or a successful conversation. This monitor can me modeled by means of an operator that takes as input an agreement and an actual conversation and replies true if the conversation is compliant with the agreement, false otherwise. Therefore, we need to provide a notion of conversation compliant with an agreement. Actually, this notion is already provided by the $\sqsubseteq$ operator. In a nutshell, a conversation is compliant with an agreement if it matches the agreement.

**Definition 3.27 (Monitor operator)** Given an agreement $\mathfrak{A}$ and a conversation $C$, the monitor operator $\circlearrowleft$ returns true if $C$ matches $A$:

$$\mathfrak{A} \circlearrowleft C = \begin{cases} true & \text{if } C \sqsubseteq A \\ false & \text{otherwise} \end{cases}$$

This is clearly an abstract function whose practical implementation is left to further research.
The monitor will then apply the $\circlearrowleft$ operator to the current conversation and the negotiate agreement. The returning result will then now be used to evaluate whether the conversation was successful or not.

#### 3.6.3.3 Direct experience evaluation

As the conversations between consumer and WS grows, the number of feedbacks returned by the monitor increases. These feedbacks can be used to build the *direct experience* evaluation towards the WS. Thus, the trustworthiness of the WS based on the direct experience depends on the amount of successful interactions with it. In other words, the trustworthiness of the web service improves if it behaves as agreed over the time.
Assuming the rate by direct experience is within a range [0, 1], we define the update function for the direct experience rating.

**Definition 3.28 (Direct Experience Rating Update)**
Given

- $\mathfrak{A}$ an agreement with a service $s$
- $C^s$ a set of new conversations from the last rate update
- $C^s_i$ the $i_{th}$ new conversation with $s$.
- $n$ the size of C
- $N$ a user defined threshold
- $R_c$ the current direct experience rate for service $s$
- $\mu$ a factor in the range [0,1]

$$Update(R_c, A, C, n) = \begin{cases} R_c + \mu * (1 - R_c) & \text{if } n = N \wedge \forall_{i \in n} \mathfrak{A} \circlearrowleft C^s_i \\ R_c & \text{if } n \neq N \wedge \forall_{i \in n} \mathfrak{A} \circlearrowleft C^s_i \\ R_c * (1 - \mu) & \text{otherwise} \end{cases}$$

The starting rating can be given either a value according to the reputation the WS have when selected, a manually selected value or a neutral 0.5. The function will keep the rating always within the range [0,1]. The rate will increase after N successful conversation, whilst it will decrease immediately if the conversation fails. Moreover, the rating decrease is faster as opposite to the increase, so eventual automatic emergency procedures based on the current value of this rating can take their effect faster.

#### 3.6.3.4 Automatic Rating

The monitoring procedure allows the consumer to keep a constantly updated rating about the interaction experience with the WS. This automatic rating system can be exploited in three ways:

- The *direct experience* ratings for each WS can be stored locally and used as an *evidence source* as well.
- By means of a *watch dog*, it should be possible to rise an alarm whenever the rating falls beyond a pre-defined threshold. This way, it is possible to initiate a recovering procedure to replace the "malfunctioning" WS with a

new one, after having queried the evidence supplier for new informations about other services.

- The ratings generated by the contract monitoring procedure can be used by the *reputation component*: they are communicated to the *trust brokers* to update the reputation of the WS. As discussed in Section 3.5.2.3, ratings should be communicated when necessary. In this case, the new rate should be communicated after the *Update* function actually updates the ratings, either by a positive or a negative increment.

Chapter 4

# Application

Throughout Chapter 2, a thorough analysis about the state of the art for SOA trust provisioning has been provided. The main issues have been highlighted and the guidelines for a new framework have been provided. In Chapter 3, the theoretical principles to found the new framework on have been defined. Essentially, the framework has been explained.
In this chapter we aim at providing a practical explanation of the effectiveness of the framework. In Section 4.1 we will provide an intuitive idea of how the framework is supposed to work. We will then highlight in section 4.2 the achieved improvements explaining how the problems arisen during the analysis have been addressed.

## 4.1 Test cases

The framework explained throughout Chapter 3 has been designed with the purpose of solving (or at least partly solving) the problems arisen and analyzed in Chapter 2. During the analysis of the state of the art, a running example has been provided. This way it was possible to exemplify the limitations ad advantages of each approach with a practical scenario.
Following the same methodology, we are now going to illustrate the impact of

our framework in a real application. As for Chapter 2, the Virtual Travel Agency use case is adopted, providing a set of scenarios along the line of those listed by W3C in *Web Services Architecture Usage Scenarios*[1]. Other scenarios that we think worth it to analyze are also described.

### 4.1.1 Assumptions

The following test cases will involve Alice as the actor representing the consumer of the Web Services. She will play the role of both a direct consumer and a developer.
Few assumptions have been made to support the test cases in this section:

- The consumer always finds a way to obtain an updated list of WS references matching his/her requirements;
- QoS offers, where mentioned, have been retrieved by means of a third party in charge of collecting and providing them. This third party can be either a central authority or a P2P network or anything else.

### 4.1.2 Background

Use cases are drawn from the W3C provided use-cases Web Services where a Virtual Travel Agency agent is considered.
An example is the scenario illustrated during the introductory Chapter 1. Moreover, a scenario involving Alice as a direct consumer is used:
*Alice, a business traveler in Copenhagen, is having dinner in a WLAN-equipped pub. Since the work meeting planned for the next week has been canceled, Alice would like to spend the unexpected free time as a tourist. Therefore, Alice connects her mobile smart phone to the WLAN network of the pub looking for some tourist Web services. With the help of her agent assistant she manages to discover many different WSs that can be used to book flights, accommodations and so on.*
With the introduction of our framework, the scenario is extended like this:
*Alice installed our framework on her smartphone. Whenever she has to choose whether to trust a Web Service or not, she will use the tools provided by the framework to gather the evidences she needs to consider the WS trustworthy.*

[1] `http://www.w3.org/TR/ws-arch-scenarios/`

### 4.1.3 QoS based selection

***Precondition:***

Alice's agent discovered a list of four[2] WSs offering flight booking capabilities. She has no previous knowledge about anyone of them.

***Case:***

Alice sets up the *Evidence Supplier* to rank them based just on her QoS preferences, then weights are distributed as:

$$\begin{aligned} \mathit{Reputation} &= 0\% \\ \mathit{QoS} &= 100\% \end{aligned}$$

No thresholds are specified.
Preferences are:

- Alice is connected to a public WLAN infrastructure. She then wishes to keep her data confidential while interacting with the Web Service. She adds **confidentiality** to the list of preferred QoS attributes.
- For efficiency reasons, Alice would like her transaction with the flight booking WS to be as quick as possible. She adds **response time** to the list of preferred attributes.

Although the efficiency suites her needs, Alice still considers the end-to-end **confidentiality** of her data the most important attribute in a public WLAN infrastructure. Based on this she sets the weights:

- $Confidentiality\ =\ 80\%$
- $ResponseTime\ =\ 20\%$

Furthermore, she sets a 100% weight on the QoS ranking and no thresholds (3.3.3).
The four WSs provide the offers illustrated in table 4.1 (non matching attributes are omitted because they weight 0, whilst missing matching attributes have the '$-$' symbol).
The values assigned to **confidentiality** derives from this assumption: *confidentiality* is a function mapping the confidentiality related parameters (i.e. encryption algorithm) to a value reflecting their quality (i.e. SSL with AES-256 encryption would gain a better value than SSL with RC4-128 encryption).
Scores are computed assuming $\gamma = 1$ in function 3.14.
The scores have been computed with the formulas specified in definitions 3.10, 3.13 and 3.14.
The resulting rank is then: **B, A, D, C**.

[2]We use a small number for the sake of simplicity

| | **Confidentiality** | **Response time** | **Score** |
|---|---|---|---|
| **A** | 3.0 | 42.0 ms | 1.528 |
| **B** | 3.5 | 25.0 ms | 1.906 |
| **C** | - | 30.0 ms | 0.275 |
| **D** | 2.0 | 10.0 ms | 1.2 |

Table 4.1: QoS Test Case

***Postcondition:***

Alice now already has a clue about which WS better fits her non functional requirements. She picks up service **B**.

***Alternative flows:***

- Even if service **B** has a promising QoS offer, Alice prefers to verify his reputation before making a choice.
- Even if service **B** has a promising QoS offer, Alice prefers to negotiate a contract to rely on directly with the provider.

### 4.1.4 Reputation based selection

***Precondition:***

Alice's agent discovered a list of four WSs offering flight booking capabilities. She has no previous knowledge about anyone of them.

***Case:***

Alice sets up the *Evidence Supplier* to rank them based just on reputation, then weights are distributed as:

$$\begin{aligned} Reputation &= 100\% \\ QoS &= 0\% \end{aligned}$$

No thresholds are specified.
As described in 2.3.2.1, the *Trust Broker* that will be queried for the reputation score requires a threshold representing the minimum number of transactions. If the threshold is not met, the WS is labeled as *unclassified.*
Alice sets up the evidence supplier so to query the *trust broker* with the following input:

$$< Alice,\ [A, B, C, D],\ 42,\ false >$$

Alice is asking for a reputation score based on at least 42 transactions for the four listed WSs. The reputation has to be calculated without considering the timestamp of the rate submissions. The *TB* then returns:

$$\begin{aligned} Classified &= [\, A = 0.4, \\ &\quad\; B = 0.2, \\ &\quad\; C = 0.9\,] \\ Unclassified &= [D] \end{aligned}$$

The result indicates that service $C$ has generally demonstrated more trustworthiness over the time, whilst $B$ probably belongs to the *malicious* category.

***Postcondition:***

Alice has now a clue about the community trust towards the four WSs. She decides that this is a sufficient quantity of evidence and she picks up **C**.

***Alternative flows:***

- Even if service **C** has a good reputation, Alice would like to know more about the non-functional offer of the Web Service.
- Even if service **C** has a good reputation, Alice prefers to negotiate a contract to rely on directly with the provider.

### 4.1.5 Aggregated sources selection

***Precondition:***

Alice's agent discovered a list of four WSs offering flight booking capabilities. She has no previous knowledge about anyone of them.

***Case:***

Alice sets up the *Evidence Supplier* to rank them based just on reputation, then weights are distributed as:

$$\begin{aligned} Reputation &= 40\% \\ QoS &= 60\% \end{aligned}$$

No thresholds are specified. The evidence supplier retrieves the required information from the two sources as described in scenario 4.1.3 and 4.1.4. It essentially computes the overall grade for the service list. First, the results are normalized to the same range of values:

$$\begin{aligned} QoSrank &= \{\ B = 1.0, \\ &\quad\ A = 0.743, \\ &\quad\ D = 0.548, \\ &\quad\ C = 0.0\ \} \\ Reputationrank &= \{\ B = 0.2, \\ &\quad\ A = 0.4, \\ &\quad\ C = 0.9\ \} \end{aligned}$$

The resulting aggregated result based on the weights is then:

$$\begin{aligned} Classified &= \{\ B = 0.68, \\ &\quad\ A = 0.605, \\ &\quad\ C = 0.36\ \} \\ Unclassified &= \{\ D\ \} \end{aligned}$$

***Postcondition:***

Alice has now a weighted rank of the WSs. She now decides to trust service **B**.

***Alternative flows:***

Alice would like to know whether she can use her VISA Electron credit card, as she usually prefers that one for online transactions. Thus, she wants to initiate a negotiation with the provider to have more details about the payment before deciding.

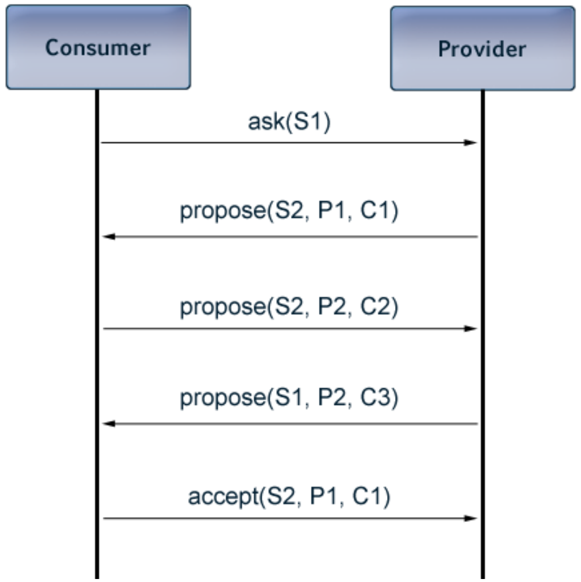
### 4.1.6 Negotiation

***Precondition:***

Alice's agent discovered a list of four WSs offering flight booking capabilities. They have been ranked by the *evidence supplier* based on Alice preferences and they are now sorted in this order: B,A,C. Service D is unclassified.

***Case:***

Alice is subscribed to a club within her bank where special gift are given after collecting enough fidelity points. Fidelity points can be collected even by doing transactions with the VISA Electron credit card. So Alice prefers to use it to book the flight. She then starts the negotiation with the provider, where she asks what privileges she has to disclose for certain services.



**S1** = selectFlight,bookFlight, confirmTransaction
**S2** = selectFlight, bookFlight, getDiscount,
confirmTransaction
**P1** = email, Master Card number
**C1** = < selectFlight ← ∅,
bookFlight ← email,
getDiscount & confirmTransaction ← Master Card number >
**P2** = email, VISA Electron number
**C2** = < selectFlight ← ∅,
bookFlight ← email,
getDiscount & confirmTransaction ← VISA Card number>
**S1** = selectFlight, bookFlight, confirmTransaction
**P2** = email, VISA Electron number
**C3** = < selectFlight ← ∅,
bookFlight ← email,
confirmTransaction ← VISA Card number>

As the negotiation flow illustrates, the flight booking service provider offers a discount for whom is going to use a Master Card. Even though Alice preferred to use a VISA Electron card, the possibility of the discount changed her mind. Then she decides to conclude the negotiation by accepting to confirm the transaction providing her Master Card number.

***Postcondition:***

Alice negotiated contract **C1** to consume the service. She is going to book the flight aware that she will obtain a discount using her Master Card. If no discount will be provided (*getDiscount* invocation will return an error), a contract violation will be caught by the monitor and the *direct experience* rating for this WS will decrease.

***Alternative flows:***

Alice is not interested in the discount, and the negotiation ends accepting contract **C3**.

### 4.1.7 Automatic feedback delivery (Alice direct consumer)

***Precondition:***

By means of the VTA software agent and the *evidence supplier*, Alice selects a flight booking WS to book her flight. The *negotiator* is then used to agree on a contract with the provider (the same contract described in 4.1.6.

***Case:***

The agent is set to monitor the conversation between Alice and the WS. After each use ($N = 1$ in formula 3.28), the rating is automatically generated by the monitoring agent and eventually sent to the *TB*.
The update is calculated according to Def. 3.28. Let us assume that $\mu$ is set to 0.2. The starting rating of the flight booking WS is set to 0.76 (drawing the value from the current reputation of the service).
Alice books the flight successfully and the security behaviour of the WS meets the agreement. The evaluation based on Alice's direct experience is updated to 0.808. The value is automatically submitted to the *TB* along with the timestamp. The *TB* updates the current reputation of the WS.

***Postcondition:***

The reputation of the flight booking WS has improved thanks to the successful conversation between Alice and the WS.

***Alternative flows:***

Alice is asked for an additional International Passport number to complete the transaction and the conversation fails (agreement not respected). The

provider was probably offering some sort of phishing feature to attract more consumers to the final step of the booking.
The current rating is updated to 0.608. The new rating is submitted to the *TB* that updates the current reputation according to the sent value.

### 4.1.8 Automatic feedback (Alice developer)

***Precondition:***

Alice has selected a flight booking WS from a list after having agreed on a contract with the provider. The agreed contract is the same resulted from the negotiation in 4.1.6. The WS joins the choreography for the VTA system. In this case, the system is going to be used by external users, unaware of the underlying service oriented architecture.

***Case:***

Alice sets a software agent to monitor the communication between the consumer (usually the code performing the WS invocations) and the provider. The agent is instructed to use the negotiated contract to verify the correctness of the interaction. The average traffic volume towards the flight booking WS is supposed to be 100 requests a day. Therefore, Alice sets the agent to update the direct experience evaluation every 50 successful conversations (twice a day). The update is calculated according to Def. 3.28. We assume that $\mu$ is set to 0.2. The starting rating of the flight booking WS is set to 0.5.
The WS performs well for 3 sequential days.
The 1st day, the rating is updated once, from 0.5 to 0.6.
The 2nd day, the rating is updated twice, from 0.6 to 0.68 to 0.744.
The 3rd day the traffic volume decrease and the rating is updated once, from 0.744 to 0.795.
For each update, the new rating is communicated automatically to the *trust broker*. The 4th day, at the 42nd conversation, the flight booking WS experiences a fault and during the *confirmTransaction* invocation, it returns an *515 - Internal server error* instead of the usual confirmation number. The rate is updated from 0.795 to 0.636 and communicated to the *TB*.

***Postcondition:***

The service performance has been tracked and rated throughout the 4 days without any human intervention. The *TB* connected to Alice's VTA has received the evaluations about the service behaviour and the reputation has been kept updated. The 4th day, after the fault, the reputation has been influenced by Alice's agent provided rate, reflecting the WS server problems without overloading the trust broker with connection for each of the 242 conversations. Only five connections have been in fact performed.

### 4.1.9 Implementation suggestions

As already explained, the framework has not been implemented in a working solution. However, many technologies have already been devised to solve some of the concerns emerging in our work. We are now going to provide some implementative examples to show how the framework could be theoretically concretised in a real application. We will just mention some ideas on how to map existing technologies to some component of the framework. Wi will not discuss about those part where the only effort is related to the coding.

#### 4.1.9.1 QoS component

There exist many standards and languages to specify QoS information regarding Web Services, in particular for what concern SOAP-based Web Services. A UDDI data structure extension named *tModel* is suggested by W3C to express QoS offers for published WS [74]. This model has been conceived to let the consumer search a UDDI registry to locate registry entries meeting a particular QoS need.
The framework does not assume the existence of a centralized UDDI registry performing the matching task, thus a more generic solution to specify QoS offers would be to adopt one of the many XML based languages for QoS specification, like OWL-S, WS-Agreement or the new OWL-Q proposed in [51]. These express generic aspects of QoS features. Other languages like WSOL, WS-QoS, DAML-QoS support class of services.

#### 4.1.9.2 Reputation component

Concerning the Reputation component, the *trust brokers* can be implemented as software agents exposing services for both retrieving the *ratings* and to update them. Those services could be exposed as Web Services as well, so to allow a sort of preliminary trustworthiness evaluation of the *trust brokers* themselves by means of our framework. Trust brokers can even provide their CA signed X.509 certificate to let the consumer make a preliminary selection.
The authors of the *distributed trust broker architecture* suggest to implement *trust brokers* using some common software package like those certified by Liberty

Alliance [1].
Anyway, a working system has been implemented by the authors themselves and performance testes have been already provided in the original work.

#### 4.1.9.3 Evidence supplier

The evidence supplier, in order to be fully flexible and extensible, should treat uniformly all the different evidence sources in the system. They could be represented with XML with the following schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<sources>
  <source name="reputation" weight="0.8" threshold="0.3">
    <classified>
      <ws ref="http://companyA.com/FlightBooking.wsdl">0.7</ws>
      <ws ref="http://companyB.com/FlightBooking.wsdl">0.5</ws>
    </classified>
    <unclassified>
      <ws ref="http://companyC.com/FlightBooking.wsdl" />
    </unclassified>
  </source>
</sources>
```

#### 4.1.9.4 Negotiation component

The negotiation can be carried out by means of software like TrustBuilder2, a work by Winslett et al.[55]. TrustBuilder2 is a fully-configurable and extensible framework for prototyping and evaluating trust negotiation systems, supporting the use of multiple credential formats such as X.509 certificates [37] and SAML assertions [18]. Furthermore, it supports negotiation strategies. It finally accepts a wide range of policy specification languages, like Cassandra [2], X-TNL [10], TPL [36], RT [58], and XACML [4], letting a great flexibility on the language choice when the implementation will be carried out.

## 4.2 Discussion

In Chapter 2, a thorough analysis regarding the current state of Web Service Trustworthiness provision has been carried out. All the existing approaches have been categorized in four main classes:

- *direct experience*
- *trusted third-party*
- *hybrid*
- *automated trust negotiation*

*TTP* based approaches and *hybrid* ones have been split in further classes (more details in Sections 2.3.2 and 2.3.3).
Each class has been discussed in relation to its advantages and shortcomings, highlighting the main limitations of the approach. At the end of the chapter the pros and cons have been categorized and mapped to each class, in order to have a starting point to devise the architecture of a new framework.
Based on the aforementioned analysis, the new framework has been conceived and described in chapter 3. According to the requirements highlighted throughout chapter 2, we are now going to trace down how each listed issue has been addressed with our framework.

### 4.2.1 New Service ramp-up

Whenever a new Web Service joins the network, it may hardly escalate his reputation in a social system where other functionally equivalent Web Services are available. In the proposed framework the issue is present when it comes to obtain an evidence through the reputation manager: in case the service is not rated enough, it will not reach the minimum threshold required by the user (if sufficiently high). As a consequence, no reputation score will be provided. Nevertheless, the user is still able to make up his trust belief in other ways: by means of the QoS matchmaker and the Negotiator. Both of them enforce the notion of *hard trust*. The potential consumer will be able to query the QoS matchmaker to verify whether the new Web Service has a valuable QoS offer and then *negotiate* a security behaviour directly with the provider. This will consequently fuel the new service rump-up, because more users will manage to consume it even when brand new.

### 4.2.2 Community dependency

Another major issue of pure community based trust systems is the community itself. A community does not born big, but starts with few, maybe inactive users. Before any kind of social help can be obtained, the society has to grow. The problem of *community bootstrap* arises in many frameworks and some of them suggest to fix the issue by providing some monitors autonomously "trying" the WSs to generate some rates. In our framework, the possibility of agreeing on a contract produces a benefit in this sense: the interaction between consumer and provider can be constantly monitored and verified, therefore both positive and negative experiences can be immediately and automatically translated in good or bad rates to communicate to the *TBs.* This will help the reputation system to become more reliable over the time, *helping the bootstrap of the community* without the need of a user manually submitting his rates (even though this is a possibility).

### 4.2.3 Unconditional trust/distrust

This problem was mainly affecting those approaches based on the direct experience. The problem in this framework is almost completely overcome: the only occasion when a user should unconditionally trust or distrust a Web Service/provider is in case no evidences are provided by any of the components interacting with the *evidence supplier* and no negotiations has been carried out successfully. This is actually an extreme situation that might be interpreted as an untrustworthiness evidence: a Web Service provider that did not apply any effort to procure either a QoS offer or a negotiation for a contract, has either such a great reputation that there are no needs for further evidences (but this is not the case because we are assuming the user does not have any evidence) or its neglecting behaviour towards the trust precautions could correctly classify the Web Service/provider as untrustworthy.

### 4.2.4 Centralized

The centralized nature of a system like the one we are discussing, would bring to the issues listed in 2.3.5. This is why our *system as been kept uncoupled by a central authority.* This does not mean that central authorities are useless and so not considered (in fact, reputation authorities are used as a last chance in the reputation infrastructure), but just that the system does not rely only on that. *Single point of failures have then been avoided* by creating more sources where to retrieve trustworthiness information from. When one of them fails, the others are still available. Even the reputation system itself has been kept distributed, by assuring that if a *TB* collapses, others can be contacted. In detail:

- Trustworthiness score is calculated in loco by the evidence supplier and the outcome of the computation can be inspected to better know how the result came up;

- The system relies mainly on three external entities: a QoS supplier (not discussed in this work), a referral in the reputation community and the negotiator agent side. This way it will be unlikely to have a global failure of the system.

- The reputation component as been outsourced from those studies that where devising a distributed architecture for the reputation community. The resulting *Reputation Agent* then inherits the features of those studies.

- Neither the consumer nor the provider are forced to disclose any sensitive information:

  – the consumer will setup the client side evidence supplier with his preferred thresholds and preferences;

  – the provider can share the QoS information related to the Web Service and keep the sensitive data protected, ready to be negotiated with the potential consumer directly;

  – an increasing number of Web Services or Web Service consumers will not destabilize the system: the service consumers have the trustworthiness mining mainly executed in their own machine. There are some possible *bottlenecks* in the system though:

    ∗ In case the reputation community is not properly setup, the amount of *trust brokers* might not satisfy the consumers' demand (in term of requests per time unit), leading to a bottleneck. We believe the issue would be overcome by a pure P2P reputation network, because the increase of consumers would lead to a proportional increase in the number of *trust brokers* (assuming each consumer would act as a trust broker too). However, as discussed in 2.3.2.1, there are a few important advantages in the current architecture of the reputation network that trade off this potential shortcoming.

    ∗ Another potential bottleneck might rise from the QoS provisioning system. As stated in chapter 3.4, this work does not include any hypothesis about how the QoS provider should be conceived and plugged to the framework. Therefore, we just point out that there might be a bottleneck in case the QoS provider will not result scalable for a service oriented environment.

### 4.2.5 User-fitting score

As we advocated in 2.3.5, the score provided by the system would be more valuable and effective if fitting the user needs, habits and personal attributes. In our framework, this is achieved in two ways:

- The result provided by QoS ranker is computed matching the user personal preferences on the QoS features of a Web Service against the QoS offers. The rank will then reflect the consumer personal needs related to the Quality of Service.
- The negotiation is executed considering the consumer provided ordered list of services, privileges and security behaviours. This will let the user specify in details how the interaction with the provider should be performed. In this case the score is 0 or 1 (either failure or acceptance), but still it is built around the user needs.

On the contrary, the reputation is computed based on the other users and trust brokers provided scores, with no details about which kind of faulty or good behaviours have generated it. Thus, the reputation score is somehow *black-boxed* and does not reflect any user's habit or need. A reputation from user **A** towards Web Service **B** almost certainly depends on the contract negotiated between **A** and **B**, that will generally be different from other users' negotiated contracts. We will discuss this issue as a potential starting step for a future extension of the framework.
Nevertheless, the reputation in our framework should be seen just as a confirmation that the WS is generally considered worth it by the society. And, moreover, that a malicious behaviour can be caught in time to the inform the other potential consumers before they start *trusting* it.

### 4.2.6 Hard to setup

This issue requires a different discussion for each of the components of the system:

- **QoS ranker**: the computation-logic of this component is kept in the client, so there are no needs to setup a specific infrastructure to support this. However there might be an integration problem arising from the QoS specification itself: to date, there are many different suggested ways to specify the QoS contract. This means that different service providers can express the QoS of their Web Services in different ways, through different standard specifications and using different metrics and parameters.

- **Reputation manager**: this component requires the setup of some initial *trusted brokers*, that can be considered trustworthy. This may result in a tough start, because the initial *trust brokers* will have to "try" the Web Services in order to mine a score. Otherwise, a consistent amount of active users is necessary in order to produce a reliable reputation score. This requirements usually affect all the *community-based/TTP* approaches, and there are no best solutions in this case, apart from trying to boost the community with an initial manual insertion: in our case, introducing some initial trusted authorities and *brokers* that can immediately provide for the reputation service.

- **Negotiator**: this component requires both the consumer and the provider to arrange an agent for performing the negotiation protocol. This is of course a strong requirement, that probably need to be addressed.

The overall evaluation is that the framework actually needs a surrounding infrastructure, even though the most part of the logic is kept on the client machine. Thus an initial effort is required to take off, but the idea is that the components can help each other in a synergistic work to achieve a stable state faster. For instance, let us assume a very poor starting community. In this case, potential consumers could rely on the QoS description and the negotiation to trust a Web Service and essentially release a reputation score. This will feed the reputation community, that will consequently allow another consumer to rely on QoS and reputation to evaluate the same service. In a nutshell, the infrastructure should theoretically grow up by an increasing speed over time. We however acknowledge that further studies about the system bootstrap would definitely improve this work.

Chapter 5

# Conclusions

In this thesis we have analyzed the state of the art for what regard trust provisioning on service oriented environment, focusing on the main issues affecting current approaches. A framework has then been suggested and motivated to overcome part of the issues arisen during the analysis. The aim of the study was to design the base framework to solve the listed problems. The design did not present any detail on the implementation or language/standards to be adopted. It has been willingly been kept abstract enough to leave free choice on its future application (no matter if it will be applied to SOAP based Web Services, RESTful ones or any others). A final test case study has been carried out to demonstrate how the framework can solve the problem listed in the analysis. Some use scenarios has been illustrated, highlighting the potentiality of the system. Moreover, a discussion about the possible application in an implemented solution has been portrayed, in order to provide a clue of the concrete applicability of the framework. Finally it has been presented a discussion about how the issues listed in the analysis have been addressed with the framework.
In this chapter we are now going to list the contributions of the whole thesis, specifying then what and how could be improved and extended in the future.

## 5.1 Contributions

### 5.1.1 State of the Art Analysis

In Chapter 2, a thorough analysis of the current state of trust provisioning approaches has been carried out. The studies addressing the problem of trust in automated systems have been analyzed and categorized based on their rationale. The categorization has highlighted the main limitations connected to each approach. A further analysis of the most critical factors influencing a trust provisioning system addressing the SOA environment has been performed, suggesting the guidelines to base a new framework upon.

### 5.1.2 Rationale for new founding principles

After the categorization based on the rationale of the approaches, a further categorization of the has been provided: those based on a *soft* notion of trust and those based on a *hard* notion of trust.
Moreover, after having emphasized the lack of clarity concerning the *trust* and *trustworthiness* term definitions in the existing literature, an informal definition of them has been provided as well.
This step allowed us to define the new founding principles to base our new framework upon.

### 5.1.3 Unified Framework

The analysis in chapter 2 highlighted the main issues related to the current trust provisioning approaches for a SOA environment. To solve part of the listed issues, a framework unifying *hard trust* and *soft trust* approaches has been devised. Thus, the most notable works based on the mentioned notions of trust have been illustrated, explaining advantages and shortcomings of each one. Some of them have been selected to join the unified framework. The *evidence supplier* component has been designed to support the collaboration of the joint components. The *negotiation* support has been provided in order to complete the framework based on its the founding principles.
The framework has been designed in light of the analysis Chapter 2 and supported by the definitions outlined after the preliminary study of the state of the art.

### 5.1.4 Referral-based Reputation Manager (extension of [61])

The distributed reputation system proposed in [61] has been extended to achieve the following results:

- parametric time-dependency for reputation computation, in order to support more use-cases
- support for multiple WSs reputation request, by extending the *trusted broker* accepted input
- integration with the evidence supplier, by extending the *trusted broker* computed result

### 5.1.5 Preferences-based QoS ranking

An accurate analysis about the QoS matching/ranking problem in literature has been carried out in Section 3.4. Out of this analysis, a new matching/ranking system has been provided. The idea results from the extensions of [97] and [104]. The new QoS ranker is based on a preference oriented demand specification, considered more usable for the end-user and more efficient. Moreover, it overcomes the issues arisen during the analysis in Section 3.4.

### 5.1.6 Contract negotiation and Monitoring (extension of [26])

The trust negotiation framework explained in [26] has been integrated in our unified framework to make it complying to the rest of the components. Furthermore, the original idea has been extended realizing a monitoring functionality to integrate to the security behaviour negotiation of [26]. This will allow to automatically verify the compliance of the communication between provider and consumer to the agreed contract, fulfilling the *credible evidence* requirement mentioned in definition 3.3. As a consequence, the user will be able to track his experience with the consumed WSs, making it available to other users and, moreover, to eventually support watch-dogs and alarms to increase the robustness of the system.

## 5.2 Future Works

Due to the time limitations, there have been some points that were not treated completely or questions that were left unanswered. We are going then to list the possible extensions and improvement that we have identified that could increase the value of this project.

- Better QoS specification: the QoS ranker relies on some user preferences expressed by means of weights. A step forward for this component would be to integrate some methodologies to allow a more semantic QoS specification, so to avoid the integration problem described in 4.2.6.

- The negotiation component has been thoroughly explained in the original paper under the point of view of the protocol and the techniques to enforce a more robust negotiation. In the same paper, the efficiency and the effectiveness of the framework is mathematically proven. We extended the framework with a monitoring feature, but this although is still missing practical credibility.

- The reputation component currently supports a really generic rating system. It would be desirable to let the user know which parts of the other users' experience with the WSs have been evaluated and which exact problems eventually arisen. This would allow a more detailed and targeted evaluation.

- Another issue related to the reputation component is the *Trust Broker* selection. Currently, the user has to manually select the Trust Broker he believes to be trustworthy and manually change it in case of faults. It would be desirable a way to keep a list of trustworthy Trust Brokers to let an agent automatically switch TB whenever a failure occurs.

- Ratings provided by different users are always considered trustworthy. Collusion is prevented just among Trust Brokers, but malicious users can still organize themselves to poison the rating of a particular service. Many works address this issue, like [39] or [62] (to mention just two of them) and they could be analyzed and revised to be integrated in our framework.

- There is currently no support for WS automatic composition. Services can join a so called *choreography*, i.e. they are orchestrated to achieve a final goal given by the composition of their results. So far, the framework supports a single-service per query evaluation. A specific infrastructure to support the non-functional automatic composition of WS is instead still missing. It would be suitable to provide the tools to evaluate not just the best service, but the optimal composition (that may contain some sub-optimal WS). Notable works in this direction are provided in [104] and [63].

Appendix A

# Publication - NODES11

# Analysis of Trust-Based Approaches for Web Service Selection

Nicola Dragoni, Nicola Miotto, Davide Papini

Department of Informatics and Mathematical Modelling
Technical University of Denmark, 2800 Kongens Lyngby, Denmark
ndra@imm.dtu.dk, s094348@student.dtu.dk, dpap@imm.dtu.dk

**Abstract.** Service Oriented Computing is an emerging paradigm for distributed computing, where Web Services represent the bricks of a Service Oriented Architecture. Brought to its full potential, this vision could allow software developers to take advantage of agents to automatically discover and compose Web Services over the Internet to build a distributed system. In the past years, there have been many issues discussed about web services, regarding their implementation, their founding principles and so on. But there is still a concern that did not gained much attention so far and would need to be thoroughly investigated: how can a service be trusted? What is the right choice when it comes to decide the best service among plenty of similar ones? In this paper we are going to show how it has been tried to answer this question, providing a survey about the state of the art of the web service trust provisioning.

## 1 Introduction

In the past years the Service Oriented paradigm has gained a growing attention, considered a new revolution in the Internet age. One of the major advantages offered by this new paradigm is the possibility of automating the process building distributed systems. A software agent should be delegated to discover the right web services (WS) to be used in a composition. This would also optimize integration and reusability of software components.Many studies have addressed the different problems related to WS and SOA, like implementation, funding principles and so on. But WS trustworthiness provision is still an open challenge. In this paper we are going to update the survey *A survey on trust-based web service provision approaches*[1] of Nicola Dragoni with new concerns and extend it with a final discussion where the different approaches found in literature will be compared, giving few suggestion on how the research on this field could proceed.

### 1.1 Sources of trust

The studies we are going to analyze throughout this paper adopt different metrics to evaluate the trustworthiness of a WS. The most part of those metrics boils down to the QoS metrics for WSs identified by W3C (figure 1).

### 1.2 Real World scenario: Virtual Tourism Agency

Alice is a software developer for a tourism company. She's asked to develop a Virtual Tourism Agency (VTA), a service helping the users throughout the steps to plan a trip; she decides to break down the system to smaller capabilities: flight booking, accommodation booking, bus ticket purchase/car rent, payment. Since there are many WSs

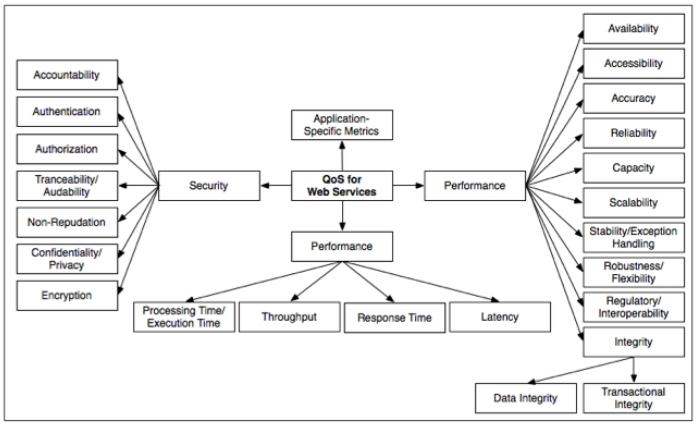


**Fig. 1.** QoS metrics according to the W3C Consortium

providing for the identified features, she has to choose the right ones and compose them in a step by step procedure; *but, which service is the* ***right*** *one*? Alice didn't use any of the available services in her career, so she picks them up just relying on few descriptions and on her common sense; after one month, the company providing for the *flight booking* WS has to temporary shut down the service because of overloading problems. Consequently, the VTA has to be taken down as well.
As the example tries to stress, selecting the right service does not include only the problem of discovering services on the basis of what a service can do (functional properties), but also how well a service can do (non-functional properties), evaluated according to some non-functional QoS[1] metrics. The QoS value may determine the trust Alice has towards a service.

## 2 Suggested approaches

There have been many studies in literature addressing the problem of *automated trust provisioning*, not necessarily directly targeting the *WS* domain. The purpose of this document is to classify the most relevant ones according to their rationale, in order to outline their advantages and limitations in a SOA environment.
The different studies can be grouped according to diagram 2 [1].
Another classification can depend on the architecture: distributed or centralized. In the following sections we will discuss the suggested solutions and their limitations, providing an example scenario where supposed useful.

### 2.1 Centralized vs Distributed

Trust provisioning systems can be built mainly around 2 architectures (apart from TN): centralized or distributed. Even if each single approach applies different rules for the

[1] Quality Of Service

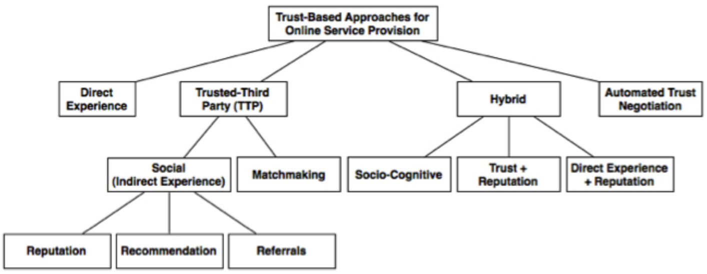

Trust-Based Approaches for Online Service Provision
Direct Experience
Trusted-Third Party (TTP)
Hybrid
Automated Trust Negotiation
Social (Indirect Experience)
Matchmaking
Socio-Cognitive
Trust + Reputation
Direct Experience + Reputation
Reputation
Recommendation
Referrals

**Fig. 2.** Classification of Trust Approaches for Online Service Provision.

entity-to-entity communication or uses different algorithms to compute the trust, the architecture-dependent behaviour can be generalized as follow:
**Definition 1 -** ***Distributed trust provisioning approaches***: *the trust scores of WSs / service providers are computed/derived after having directly communicated with other peers in the system.*
**Limitations**: in the context of SOA is not really possible to generalize the shortcomings deriving from a distributed architecture, because they are tightly connected to the specific approach. Anyway, in general, a common drawback is the system setup and start-up effort.
**Definition 2 -** ***Centralized trust provisioning approaches***: *the trust scores of WSs / service providers are provided by a central authority with the responsibility of computing/collecting them.*
**Limitations**: Central authorities are a single point of failure and thus can exist only under rigidly constructed and administered computational environments, in particular considering the capacity demand of a SOA environment. Another technical limitation resides in the possible alteration of the ratings (collusion or retaliation). Moreover, a centralized trust authority *can never be a good enough recommender for everyone: different entities should be allowed to make up their own mind*[2].

Further advantages and drawbacks will be discussed in the following sections according to each approach.

### 2.2 Direct Experience

This class of approaches is based on *presumptions drawn from the service consumer's own direct experience with the target service* [3]. The rationale is that the trust can be build upon some quality parameters that depend on the service behaviour in the course of time. This means that the service can be trusted if the consumer past experience with that service, i.e. *the knowledge gained after having a transaction with it*[3], results complying to his expectations and requirements. Jonker and Jan Treur present an analysis of models for the dynamics of trust based on experiences [4]. They investigate the basing principles governing the evolution of the trust towards a service. In [5] the authors describe a layered framework conceived to manage trustworthiness through seven levels. As the authors point out *the quality of the model to be built is fully dependent on the experience of the practitioners*. In other words, how to trust a service when no past

experience is available? This is the main drawback of the trust-by-direct-experience approaches.
**Definition 3 -** ***Trust by Direct Experience***: *a service consumer trusts a service because of his good past experience with the service.*
**Limitations:** This approach is not suitable for large open systems where anyone can publish its (malicious) code, since it does not allow to trust a service before its execution. Moreover, whenever an *unconditional distrust* approach is used, brand new services may be not considered even if conforming to the needs.
**Scenario 1:** Alice has no past experience with any flight booking WS. For each discovered WS, with no evidence of its trustworthiness, she is forced to either unconditionally trust or distrust it. In the first case Alice has to accept not only the WS inherent "risk of prior performance" (i.e., to pay for services and goods before receiving them) but also the "risk of blind (i.e., untrusted) execution". In the second case she is going to reject a service that might have been compliant to her trust policies.

## 2.3 Trusted Third-Party Approaches

TTP approaches are based on the idea that the service consumer can rely on a third-party in order to obtain a trust value of a given service. *Third-party* may refer either to a trusted central authority or members of a community. The underlying assumption of these approaches is that consumers must trust the third party they decide to consult. We distinguish among two types of approaches: social and matchmaking approaches. In both of them the final decision is based on the assessments provided by the TTP. The difference lies in how the assessments are computed.

**Social (Indirect Experience)** The trust evaluation towards a WS is forged by a cooperating community whose members have directly or indirectly interacted with such WS. In order to be effective, each community member has to continuously review the services (and the service providers) it's using.The global evaluation is not necessarily calculated by the community members themselves, but might be the result of a centralized data mining applied on member-supplied informations. In literature we can find three different social-based approaches: *reputation*, *recommendation* and *referral*.
***-Reputation:*** The definition of the term *reputation* from The English Oxford Dictionary is: "*a widespread belief that someone or something has a particular characteristic"*. In a SOA context, the rationale is that a WS is trustworthy as long as the community has a good opinion about it. The reputation system is responsible to collect ratings about users, services and service providers from members in the community. The global opinion can be modeled either around the QoS parameters described above or depending to what degree WSs adhere to the contract. However, the parameters used to rate a service, user or service provider are not influencing the rationale of a TTP reputation system: an individual's subjective trust on a service is derived from the reputation of that service or, in other words, from the direct experience of someone else.
**Definition 4 -** ***Trust by Reputation***: *a service consumer trusts a service because of his good reputation.*
A major distinction between different reputation systems is outlined by the base architecture: centralized or distributed [6][7]. Most of the studies suggests centralized

approaches[8], typical of e-commerce web sites[2]. In those systems a central authority is responsible to collect all the ratings from other members in the community (e.g. QoS data from WS consumers, in our case) who have had direct experience with a specific service or provider. The authority uses these ratings to derive a reputation score for the service and makes it publicly available to future, potential consumers. Regarding distributed reputation systems, two notable examples are EigenTrust [9] and the PeerTrust [10]. Each member of the community (be it an agent or a human) records its own opinion about a service to make it available to the others. A reputation grade is a function of all the trust ratings (if there are any) obtainable by all the possible members the agent can reach.
**Limitation A**: the effectiveness of any reputation system lies on the number of members in a community and on their behavior. The fewer the members in a reputation system, the more inadequate the ratings provided by the systems. This issue envelopes the *community-bootstrap* problem:
**Definition 5 - *Community bootstrap issue***: a community-dependent system is unlikely to provide good quality results as long as the community is small or not really active.
**Limitation B**: another shortcoming is that trust relies on past information from other members of the community. A natural problem arises in case of new services. For example, when a service initially registers for business, no other consumer has interacted with it and consequently no information exists about its past behaviour and questions about its trustworthiness are left unanswered. This can be defined as the *new WS ramp-up issue*:
**Definition 6 - *New WS ramp-up issue***: a new web service takes time before being adequately evaluated.
**Scenario 2**: Alice needs a trust score for a just discovered flight booking service. She sets up an agent to query all the neighbour agents for a reputation grade. The agents (based on theirs and their neighbours past experience) return very low scores to Alice's agent. The WS is then discarded, but maybe either the community or the WS are new born and it comes difficult to provide a useful evaluation. Thus, Alice still cannot adequately evaluate the WS trustworthiness.

Now, a question for this and other social approaches arises: how the community should repute a *new service*? There have been many studies addressing this issue. In the Sporas system suggested by Giorgios Zacharia et al.,*new users start with the minimum reputation value.*[11]. The authors of the Dirichlet algorithm[7] (conceived for P2P sharing network) state that *it is possible to track the average reputation score of the whole community, and this can be used to set the base rate for new agents*[7].
**Limitation**: in general, when the starting reputation is low, the new WS is underestimated. Whenever a new WS receives an initial reputation score higher than the minimum, this can be exploited by malicious users by continuously subscribing and unsubscribing to the system in order to keep having a "non zero" reputation value.
**Scenario 3**: Alice's software agent starts discovering new flight booking WSs on behalf of Alice. It queries a TTP that states that the chosen service has a non zero grade. Alice still doesn't know whether the service is trustworthy: it might belong to a malicious provider just subscribed to the community.

[2] e.g. `http://www.ebay.com`

***-Recommendation:*** Recommendation systems [12][2] aim at making a prediction of a consumer's needs of interests. In its common formulation [13], the recommendation problem is reduced to the problem of estimating ratings for the items (such as services) that have not been seen by a consumer. Intuitively, this estimation is usually based on the ratings given by this consumer to other items or on the ratings that similar users provided for the targeted items. Once it is possible to estimate ratings for the yet unrated items, then the system can recommend to the user the items with the highest estimated ratings.
**Definition 7 - *Trust by Recommendation***: *a service consumer trusts a service because of some recommendations got from a trusted authority.*
In general, *recommendation-based systems* work as good as wide and rich the knowledge of the system is. In other words, it is necessary to know both the community and the user requesting the service in order to produce reasonable evaluations. These systems can be classified into five categories according to [1][14][12]:
in *content-based filtering*, the only static approach among the five ones, items are selected according to their content. But this approach is very primitive and would be a step backward from current WSs standards (which involve formal structured description of services); the most widely used one in e-commerce sites[3] is the *Collaborative Filtering* (CF): the consumer is recommended items that people with similar tastes and preferences liked in the past. In CF, the implicit assumption is that different people have different tastes. Note that this represents the key difference with respect to reputation systems[15]. Items recommended to the user are then the ones other users with similar tastes (neighbours) liked [16]. The next three approaches are *utility-based*[17], *demographic*[18][19] and *knowledge-based*[20][21] recommenders, where basically, as for *CF*, by means of different classification and data mining algorithms, they infer the relationship between user need/profiles and items in the community (e.g. in our case they may be WSs and other community members). The last group is identified by the *hybrid* recommenders, whose rationale is *to combine two or more recommendation techniques* (usually along with *CF*) *to gain better performance with fewer of the drawbacks of any individual one*.[14]
**Limitations**: those typologies of recommendation system share one main weakness: the system needs many information about the users in order to provide useful evaluation. This can be achieved by either asking the users to disclose maybe sensitive information (as we have seen this is not suitable in a SOA environment) or by mining them out of the interaction of the users with the system, that would require a long time. This leads to the following definition:
**Definition 8 - *New User ramp-up issue***: *a new user needs to interact with the trust provisioning system in order to receive good quality results.*
Moreover, the well known issues of the social systems are still present: community bootstrap and service ramp-up (mitigated in the hybrid recommender). Finally, the recommendation systems are conceptually centralized (see section 2.1).
**Scenario 4**: Alice finds a WS recommender service that requires to input some informations regarding the company past experience with other web services in order to be used. Alice's company does not want to disclose this information to a central system. A

[3] like Amazon `http://www.amazon.com`

second recommender service does not demand any pre-use information, but it requires Alice's agent to interact with the community for a period before being able to provide a recommendation and so Alice can't use it.
***-Referral:*** Referrals [22][23] have been proposed as a decentralized approach based on online communities and software agents technologies. An online community is a set of interacting members (or principals in the jargon) representing people, businesses or other organizations. The members of a community provide services as well as referrals for services to each other. Referrals may be provided proactively or in response to requests. Members are assisted by software agents to help them manage their interactions[22]. Referrals are based on a representation of how much the other available parties can be trusted. Agents are responsible to build and manage these representations taking into account the previous experiences of their members and communicate with each others. Participating on behalf of different members, agents appear as autonomous and heterogeneous. Moreover, agents organize themselves into communities and agents in the same community are called neighbours. A key difference from the recommendation systems is that in referral systems the participants reveal their ratings to those whom they trust, so the ratings would be more likely to be honest.
**Definition 9 - *Trust by Referrals***: *A service consumer trusts a service because of some referrals obtained from trusted software agents.*
**Limitations:** Referral systems address some limitations of reputation and recommendation systems (such as, their centralized nature) but still rely on the judgments of the members of a community and new WS are difficult to start-up. Moreover some technical practical issues, such as agents and members registration and communication as well as referrals representation, are left unanswered in the literature, making the impression of a still immature (or at least just academic) approach.
**Scenario 5**: refer to scenario **1** and **2**.

**Matchmaking** These approaches are based on a component called "matchmaker" responsible to match a user's request and trust preferences with available online service descriptions. If some matches are found than the results are sent back to the user.
A centralized trust-based matchmaking methodology has been proposed by Galizia et al. in [24]. Differently to other approaches, they embodied the WS selection problem in a classification problem: given a set of user and WS policies and established a classification criterion, the goal is to identify a class of WSs matching with trust policies of involved users. In other words, WSs are classified according to the specific user as well as trust policies.
**Definition 10 - *Trust by Matchmaking***: *A service consumer trusts a service because a trusted (central/distributed) matchmaker states that the service's policy matches the consumer's request.*
**Limitation A**: matchmakers suffer of all the drawbacks inherited by the centralized architecture (see 2.1). Moreover, both consumers and providers has to register to the matchmaker in order to use it (far from the SOC vision). Finally, it is not realistic to ask the providers to disclose all their (maybe sensitive) policies to a central authority.

Olmedilla et al. [25] replaces the centralized matchmaker and registry with a Peer-to-Peer network, distributing the matchmaking process to the service providers. A similar approach has been proposed by Olmedilla et al. in [25]. The main difference with

respect to [24] lies in the underlying registry and matchmaking architecture, which is based on a Peer-to-Peer network. Whenever a new service provider wants to offer its services, it must just join such network. On the client side, a user looking for a service must send a query along with his policies to a reasoning agent he trusts. The agent distributes the query to the peers on the network and each one of them applies a matching algorithm. Whenever a peer has matches, it sends them back to the reasoning agent which joins the results and presents them to the user. This way servers can keep policies locally and private. However the problem is moved from trusting a service or a service provider to the one of finding such a trusted reasoning agent.
**Scenario 6**: Alice, looking for a distributed matchmaking service, has to face the problem of selecting and trusting a reasoning agent. But where can she locate such computational entities? And which agents should she trust? On which basis? Current distributed matchmaking technologies do not answer, leaving Alice in the same (vulnerable) situation.
**Limitation B**: this family of approaches seems to solve the community-bootstrap problem. The trustworthiness score is evaluated matching the user trust requirements directly against the WS provided trust guarantees. However this approach exposes another issue: it wouldn't be difficult for a malicious (or distracted) user to craft a WS description so to pretend to be a trustworthy WS. In this case a "watching" community would turn useful.
**Scenario 7**: Alice's submitted the company's trust profile to either a distributed or centralized matchmaker. Now a software agent is instructed to trust WSs providing AES encryption algorithm and capable of handling 20 simultaneous connections. The matchmaker returns to the agent the matching WSs and Alice selects one. During the pre-easter week (when many people use the VTA system), the system starts having many faults. Alice realizes that the *flight booking* WS cannot handle more than 10 connections simultaneously and the VTA system has to be temporary taken down.

## 2.4 Hybrid

Hybrid approaches for trust-based online service selection are based on a combination of well known trust methodologies, improving the quality of the assessments.

**Socio-Cognitive** These approaches are mostly based on the works of Falcone and Castelfranchi [26][27][28]. Influenced by the Artificial Intelligence (AI) field and especially by the Multi-Agent System (MAS) paradigm, they treat trust as an agent's mental state. The agent supports beliefs from which is possible to derive a degree of trust. As pointed out in [3], beliefs can be seen as the answers to the question "What do we have in mind when we trust a service?". According to these beliefs the agent can articulate assumptions and expectations about a specific service.
**Definition 11 - *Socio-Cognitive Trust***: *The degree of trust is a function of the subjective certainty of the pertinent beliefs. Therefore, A service consumer trusts a service because of some of its subjective beliefs.*
The trust level is a function of such subjective beliefs. A key question therefore arises: how are such beliefs obtained? That is, from which sources? The answer to the above question is different in the various proposals in literature. Two of the most common sources of belief are the ones already discussed in the previous sections, i.e. *direct experience* (2.2) and *reputation* (2.3). In addition there are *categorization* (the process of

grouping things based on prototypes) and *reasoning* (the act of using reason to derive a conclusion from certain premises). For instance, in [4][29] the authors propose models in which they consider the direct interaction or reputation as sources. In [28] sources are categorization and reasoning. In [3] Ali et al. restrict sources to direct experience and reputation.
**Limitations:** a first weakness of the approach lies in the fact that it is based on beliefs obtained by means of the well known (and problematic) methodologies for trust. Another major limitation lies at the implementation level. To fully realize this approach, some sort of BDI[4] agents [30] is needed. Indeed, as Falcone et al. remark in their paper [26] only a cognitive agent can "trust" another agent. We mean: only an agent endowed with goals and beliefs. This requirement seems too strong when applied to open and large service-based systems, since it is not reasonable to assume that every agent will be conformed to the BDI model (which, a part from the modeling of trust, requires specific architectures to support the reasoning on beliefs and goals).
**Scenario 8**: Alice decides to delegate the trustworthy *flight booking* WS discovery to a BDI agent. Considering the openness of the system Alice is dealing with she realizes that her agent won't be capable of communicating with many other agents of the same kind. The selected WSs are then narrowed to a small amount compared to the total. The chosen WS is thus going to be far away from the best available choice.

**Trust & Reputation** Studies such as [31][32][33] propose methods for assessing the quality of online services by combining trust and reputation techniques in a single integrated framework. For instance, [33] shows how (Bayesian) reputation systems can be combined with trust modeling based on subjective logic [34]. [31] describes a report-driven framework. WSs have their QoS profile computed by means of reports provided by both providers and consumers. The profiles are used to generate a WS rank based on the consumer requirements and reputation. Then, in order to prevent "spammers" or distracted users from poisoning the system, the authors suggest, along with the reputation framework, a trust system capable of identifying *liars*.
**Definition 12 - *Trust & Reputation based system***: *A system providing for a trustworthiness score employing methodologies based on both reputation and trust, in order to improve some weaknesses of the constituent methodologies*.
**Limitations**: Although these approaches are remarkable, especially [33] where the integration results in a flexible framework for online trust management, they still suffer the main limitations of their constituent methodologies. For instance, both approaches inherit one of the main weaknesses of some reputation systems (social and centralized) (section 2.3). The authors of [33] propose a bootstrapping method consisting of creating trusted reports for the most important WSs by means of trusted monitoring agents. However, with this approach there would be the problem of selecting the most important WSs.
**Scenario 9**: refer to scenario **1** and **2**.

**Direct Experience & Reputation** [35][36][37] propose a model where the trust in a service is computed as a rating of the level of performance of the service. This overall performance is not limited to the agent's direct experience (or confidence, see section

[4] Belief-Desire-Intention

2.3) but it is also based on the evaluations of the service by other agents in the system (in [35] called the "group experience" , i.e., what the other members of the group think about the agent being evaluated and his group). Thus, in these models trust can be seen as a rating built as a result from combining agent's direct experience (with the service) along with the social reputation of the service provider.
**Definition 13 -** ***Trust by Direct Experience & Reputation***: *The trust towards a service is evaluated by means of the user direct experience combined with the service reputation.*
**Limitations**: again, the combination of two methodologies improves some weaknesses of one constituent model, but it does not provide a complete solution to the trustworthy online service selection problem. For instance, in [36] the authors combine confidence and reputation to address the situation where no previous experience of the service is available (main weakness of the direct experience method). But to do this they based their proposal on trust and reputation mechanisms to infer expectations of future providers' behavior from past experiences in similar situations. This idea inherits the already discussed problems of trust and reputation mechanisms.
**Scenario 10**: Alice's agent can't establish a reliable trustworthiness score for certain flight booking WSs because there are no past interactions with them and, moreover, they seem to have joined the WS network too recently in order to have some useful reputation evaluations.

## 2.5 Automated Trust Negotiation

Automated Trust Negotiation (TN) [38] is an approach specifically targeted to allow agents to access sensitive data and services in open environments. Trust negotiation protocols are based on the iterative disclosure of digital credentials and requests for credentials between two unknown parties (strangers in TN jargon), with the goal of establishing sufficient mutual trust so that the parties can complete a transaction. Informally, digital credentials (credentials for short) refer to the online analogues of paper credentials (a drivers license, passport, or employee ID card, for example). Thus, a credential is a digitally signed assertion by a credential issuer about the credential owner. It is usually signed using the issuers private key and verified using the issuers public key [39]. To automate trust negotiation, each party must establish access control policies (policies for short) to protect its sensitive resources, including credentials and services, from inappropriate access. Each policy should specify the digital credentials strangers must present to access the protected resource. Policies can themselves be seen as sensitive resources. Considering that both the consumer and the provider can provide their own policies to gradually disclose, the point of view is not restricted to the service consumer only anymore (how the service consumer may trust a service): the goal now is to establish a mutual trust between service consumer and provider.
**Definition 14 -** ***Credential-Based Trust (or Trust by Negotiation)***: *A service consumer and a service provider mutually trust each other because the access control policy of the requested service is compliant with the access control policy of the service consumer.*
Note that the above definition does not state that a negotiation will always succeed if the parties' policies are compliant. Indeed, the success of the negotiation depends on several factors. For instance, a negotiation could take different routes according to the negotiation strategies adopted by the parties [40]. The above definition just states that

if a trust negotiation succeeds establishing a mutual trust among two parties then this is because the two parties have compliant access control policies for the requested resource.

**Limitations**: trust negotiation principles and systems have been widely investigated in the last few years, both in different (still mainly academic) domains (like e-Business, e-Commerce, P2P systems and more recently in WSs [41]) and with respect to issues such as privacy, safety and efficiency. This effort is evident in the growing literature on TN related issues ( [39][42][43][44][45] to mention only a few). However, several key issues have still to be addressed to bring TN to its full potential: to date, the proposed frameworks seems to have been studied in theoretical and academic fashion, still "unplugged" from the real nature of WS and SOA: first of all, many in the literature treated WSs (WS) as a set of independent single operations, while interacting with real world WSs involves generally a sequence of invocations of several of their operations [46]; then SOAP-based WSs are supposed to be consumed many times from the same costumer, since they are involved in a composition. This means that a WS should be trusted the first time during discovery (development) and then be consumed without the TN protocol being involved on the ensuing requests. Moreover, no standard protocols or languages have been defined, so the different proof of concept systems are unable to talk each other. Finally, adopting a TN approach would require that both parties reason and act according to a credential-based notion of trust. Other trust meanings are not supported. A first preliminary work on this direction has been proposed by Dragoni et al. in [41][47].

**Scenario 11**: Alice's agent finds an interesting *flight booking* WS. It starts the TN protocol, disclosing step by step the required company credentials and finally it trusts it. Now the WS is inserted in the VTA. Whenever a VTA client uses the service, the *flight booking* WS requires the whole TN protocol to start over, that is not inefficient and useless because the service has already been trusted by Alice's agent during the discovery.

## 3 Discussions and conclusions

As we verified in the previous section, there have been suggested many approaches for the automatic trust provision. Some of them are not directly addressed to a SOA environment, but they can still be adapted to it. The literature about that is growing, but, as it is possible to verify out of this analysis, WS trustworthiness provision is still an open challenge.

While investigating on studies in literature, it has been possible to derive shortcomings and advantages of the single approaches. They are summarized in the table 1. In order to optimize space and improve clarity, the plus and minuses are synthesized in few main classes, each one with its notation:

**Shortcomings**

*NSR* (i.e. *New WS Ramp-up*): refer to definition 6;

*CD* (i.e. *Community Dependent*): a community dependent system is affected by the community bootstrap issue (refer to definition 5);

*NUR* (i.e. *New User Ramp-up*): refer to definition 8;

*HS* (i.e. *Hard Setup*): this problem is connected to those approaches that require a big effort to be integrated in the real world;

*UT* (i.e. *Unconditional Trust/Distrust*): this issue is related to the approaches were the

user has to consume a service without any previous experience or evidence that the service is trustworthy. Or, the other way around, the user distrust the service unconditionally for the same reason;
*CE* (i.e. *Centralized*): refer to section 2.1;
**Advantages**
*PUTS* (i.e. *Pre Use Trust Score*): there are chances to obtain a trust score *before* using the service;
*UFS* (i.e. *User Fitting Score*): the WS trust score is also somehow related to the user personal "tastes" and habits.

There are also other specific pluses and minuses related to specific approaches that will be described directly in the table.

| **Approach** | | | **Pluses** | **Minuses** |
|---|---|---|---|---|
| **Direct experience** | | | UFS → the **most** fitting score | UT |
| **TTP** | Social | Reputation | PUTS | NSR; CD; CE for those methodologies based on a centralized architecture |
| | | Recommendation | PUTS; UFS | CD; NSR; NUR or user (sensitive) information disclosure; CE for those methodologies based on a centralized architecture |
| | | Referrals | PUTS; rates coming from trusted peers | NSR; CD |
| | Matchmaker | | PUTS; UFS; some community based methodologies provide *liars* recognition | HS; CE for those methodologies based on a centralized architecture; trust towards service moved to trust towards agent in those based on a distributed architecture |
| **Hybrid** | Socio-Cognitive | | Accurate trust computation; UFS | depending on the belief source UT/NSR/CD/CE/NUR; HS because cognitive agents have to be conforming to a model |
| | Trust & Reputation | | *liars* recognition (some of the ideas); PUTS; some sort of good results can be provided even with a poor community or a brand new service | CE; NSR; CD |
| | DE & Reputation | | Issues of the 2 constituent models mitigated | NSR; CD |
| **Automated Trust Negotiation** | | | UFS; PUTS; trust can ALWAYS be computed | HS; no standards defined; no fully WS aware (at the current state) |

**Table 1.** Pluses and Minuses summary

In order to carry out this evaluation, few questions have been adopted as guidelines to judge each system:
***How does the trust score fit the user needs?*** It would be better to build the trust score around the user profile, as, for instance, the recommendation systems (2.3) do;
***Does the provider/consumer have to disclose any sensitive informations?*** Some centralized approaches ask the user (be it either the provider, the consumer or both) to submit some personal information in order to improve the trust score computation. This is clearly something that should be avoided: users usually don't want to unconditionally disclose sensitive details to a central authority.
***Can the user know how the trust is calculated?*** Depending on the system architecture, the trust score might be calculated by a third party in a black box. The user may rather prefer to know how the service he is going to trust has been suggested.
***How does the community influence the trust score?*** This issue mainly affects the *social* approaches (2.3): is the trust score depending on the size/quality of a community? In such a case, the main shortcoming would be the community bootstrap, i.e. how to create an initial community to kick-off the system.

***Does the user has to unconditionally trust/distrust certain services?*** Whenever a user finds a WS, if the functional contract meets the user needs, there should always be a way to provide a trust score for that service, without leaving the user in the position of unconditionally trusting/distrusting the WS.
***What is the trustworthiness of a brand new WS?*** A new WS (i.e. it has been recently deployed) needs a way to be "tried" even with no previous knowledge about it. The TN approaches seems to be a good starting point to address this issue.
***How hard is the trust provisioning infrastructure to setup and maintain?*** When designing a trust provisioning system it has to be taken into account both the effort needed to apply the system to the already existing SOC infrastructure, and the issues intrinsically related to the nature of a SOA-based system. A trust provisioning system is in charge of a great responsibility, and its robustness and scalability is a critical point in a SOA environment. Thus, for instance, it would be rather irresponsible to adopt a pure centralized architecture (that would be the single point of failure).

The analyzed approaches can be actually further on generalized to two big classes. The first one comprises the vast majority, i.e. the ones based either on *direct experience* of the consumer with the service, *indirect experience* (opinions on the service coming from someone trusted by the consumer) or a combination of them (*hybrid*). All of them has two key limitations in common: the user has to obtain *trust* from his own direct experience OR from the direct experience of someone else he trusts. It is usually safer to trust, for instance, 15 people saying that something is good instead of directly trusting something hoping it will be good. But still, this requires 15 people taking the risk to *try*. As correctly pointed out by Dragoni in [1]: *if someone does not take the risk of invoking an unknown service for the first time, then no one will be able to decide about the trustworthiness of the service before its invocation.* This class of approaches is based on a "soft trust" mechanism ( similar to the idea of "soft security" coined by Rasmusson et al. [48]) and they share the critical issue of service and community bootstrap. The rationale of the "soft trust" is that participants in a market collaborate each other in sharing information on other participants or services. Soft trust expect and even accept that there might be malicious services or service providers in the system. The idea is to identify them and prevent them from harming the other participants by means of collaboration and social interactions.
The other class of approaches, such like TN and Matchmakers, relies on a "hard" notion of *trust*: trustworthiness of a WS could be derived just from the a non-functional contract. They take into account the *semantic* of a WS, i.e. their security behaviour (e.g. access control rules, QoS features and so on). The recent Security-By-Contract (SxC) approach [41] might represent a good starting point for this purpose, because it takes into account the security behavior of a service instead of depending on the social control philosophy in the existing trust based approaches. Nevertheless, even the "hard trust" provisioning approaches studied in literature has a critical drawback: the lack of "fault recognition" capability. Everyone can provide a fake/wrong contract, be it due either to a malicious behaviour or to human distraction (or other unforeseeable reasons). In this case the community help would turn useful. Moreover, it seems that these kind of approaches are still studied in a too theoretical fashion, without considering the practical issues related to WSs. For instance, the TN approaches seems to work fine assuming

that the WS is consumed by a user directly invoking it. But in the real world, as said in section 2.5, WSs are a “developer thing”. This means that they have to be trusted during the development time and then transparently consumed by clients unaware of the distributed nature of the system they are using.
Hybrid systems turned to be generally improving the constituent methods. Thus, a good direction to follow is probably to design a system capable of providing the features of both the macro-families: a “soft trust” system working along with a “hard trust” one should lead to a framework where the SOA developer can always evaluate *automatically* the available WSs for a given need and where a community is able to push unworthy WSs (and providers) aside. Finally, one of the major point of confusion concerning the current discussions about trust/trustworthiness in SOA is related to the meaning of terms. The studies analyzed in section 2, often use the word *trust* and the word *trustworthiness* with the same acceptation or with different meanings in different documents. The two terms have a precise meaning and *trust* should not be confused with *trustworthiness*. This should be probably the first issue to address starting to design an acceptable solution.

## References

1. Dragoni, N.: A survey on trust-based web service provision approaches. Technical report, In Proc. of the 3rd International Conference on Dependability (DEPEND 2010), Venice, Italy, 2010, IEEE CPS
2. Abdul-Rahman, A., Hailes, S.: Using recommendations for managing trust in distributed systems. In: IEEE Malaysia International Conference on Communication. (November 1997)
3. Ali, A.S., Ludwig, S.A., Rana, O.F.: A cognitive trust-based approach for web service discovery and selection. Technical report, Department of Computer Science Cardiff University, UK (2010)
4. Jonker, C.M., Treur, J.: Formal analysis of models for the dynamics of trust based on experiences. Technical report, Department of Computer Science Cardiff University, UK (1999)
5. Zhang, J., Zhang, L.J., Chung, J.Y.: Ws-trustworthy: A framework for web services centered trustworthy computing. In: Proceedings of the 2004 IEEE International Conference on Services Computing, Washington, DC, USA, IEEE Computer Society (2004) 186–193
6. Wang, Y., Vassileva, J.: Toward trust and reputation based web service selection: A survey (2007)
7. Jøsang, A., Haller, J.: Dirichlet reputation systems. In: INTERNATIONAL CONFERENCE ON AVAILABILITY, RELIABILITY AND SECURITY, IEEE Computer Society (2007) 112–119
8. Manikrao, U.S.: Dynamic selection of web services with recommendation system. In: In: Proceedings of the International Conference on Next Generation Web Services Practices (NWESP), IEEE Computer Society, Press (2005) 117
9. Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The eigentrust algorithm for reputation management in p2p networks. In: Proceedings of the 12th international conference on World Wide Web. WWW ’03, New York, NY, USA, ACM (2003) 640–651
10. Xiong, L., Liu, L.: Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING **16** (2004) 843–857
11. Zacharia, G., Moukas, A., Maes, P.: Collaborative reputation mechanisms in electronic marketplaces. In: Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences-Volume 8 - Volume 8. HICSS ’99, Washington, DC, USA, IEEE Computer Society (1999) 8026–

12. Balabanović, M., Shoham, Y.: Fab: content-based, collaborative recommendation. Commun. ACM **40** (March 1997) 66–72
13. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING **17**(6) (2005) 734–749
14. Burke, R.: Hybrid recommender systems: Survey and experiments. User Modeling and User-Adapted Interaction **12** (November 2002) 331–370
15. Jøsang, A., Ismail, R., Boyd, C.: A survey of trust and reputation systems for online service provision. Decis. Support Syst. **43** (March 2007) 618–644
16. Huhns, M.N., Singh, M.P.: Service-oriented computing: Key concepts and principles. IEEE Internet Computing **9** (January 2005) 75–81
17. Guttman, R.H., Moukas, A.G., Maes, P.: Agent-mediated electronic commerce: a survey. Knowl. Eng. Rev. **13** (July 1998) 147–159
18. Rich, E.: Readings in intelligent user interfaces. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998) 329–342
19. Pazzani, M.J.: A framework for collaborative, content-based and demographic filtering. Artif. Intell. Rev. **13** (December 1999) 393–408
20. Towle, B., Quinn, C.: Knowledge based recommender systems using explicit user models. In: Papers from the AAAI Workshop, AAAI Technical Report WS-00-04, Menlo Park, CA: AAAI Press (2000) 74–77
21. Schafer, J.B., Konstan, J., Riedi, J.: Recommender systems in e-commerce. In: Proceedings of the 1st ACM conference on Electronic commerce. EC '99, New York, NY, USA, ACM (1999) 158–166
22. Singh, M.P., Yu, B., Venkatraman, M.: Community-based service location. Commun. ACM **44** (April 2001) 49–54
23. Yolum, P., Singh, M.P.: Engineering self-organizing referral networks for trustworthy service selection. IEEE Transactions on Systems, Man, and Cybernetics. Part A **35**(3) (2005) 396–407
24. Galizia, S., Gugliotta, A., Domingue, J.: A trust based methodology for web service selection. In: Proceedings of the International Conference on Semantic Computing, Washington, DC, USA, IEEE Computer Society (2007) 193–200
25. Olmedilla, D., Lara, R., Polleres, A., Lausen, H.: Trust negotiation for semantic web services. In: 1ST INTERNATIONAL WORKSHOP ON SEMANTIC WEB SERVICES AND WEB PROCESS COMPOSITION IN CONJUNCTION WITH THE 2004 IEEE INTERNATIONAL CONFERENCE ON WEB SERVICES, Springer (2004) 81–95
26. Castelfranchi, C., Falcone, R.: Principles of trust for mas: Cognitive anatomy, social importance, and quantification. In: Proceedings of the 3rd International Conference on Multi Agent Systems. ICMAS '98, Washington, DC, USA, IEEE Computer Society (1998) 72–
27. C., C., Y.H., T.: Trust and Deception in Virtual Societies. Kluwer Academic Publishers (pres)
28. R., F., G., P., C., C.: A - Fuzzy approach to a belief-based trust computation. In: Special issue on 'Trust, Reputation and Security: Theories and Practice'. (2003) 73–86
29. Schillo, M., Funk, P., Rovatsos, M.: Who can you trust: Dealing with deception. In: Proceedings of Autonomous Agents '99 Workshop on "Deception, Fraud, and Trust in Agent Societies", Seattle, USA (May 1999) 81–94
30. Rao, A.S., Georgeff, M.P.: Bdi agents: From theory to practice. In: IN PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON MULTI-AGENT SYSTEMS (ICMAS-95. (1995) 312–319
31. Vu, L.H., Hauswirth, M., Aberer, K.: Qos-based service selection and ranking with trust and reputation management. In: in Proceedings of the Cooperative Information System Conference (CoopIS05. (2005) 446–483

32. Huynh, D., Jennings, N.R., Shadbolt, N.R.: Developing an integrated trust and reputation model for open multi-agent systems. (2004) 65–74
33. Jøsang, A., Bhuiyan, T., Xu, Y., Cox, C.: Combining trust and reputation management for web-based services. In: Proceedings of the 5th international conference on Trust, Privacy and Security in Digital Business. TrustBus '08, Berlin, Heidelberg, Springer-Verlag (2008) 90–99
34. Jøsang, A.: A logic for uncertain probabilities. Int. J. Uncertain. Fuzziness Knowl.-Based Syst. **9** (June 2001) 279–311
35. Sabater, J., Sierra, C.: Regret: reputation in gregarious societies. In: Proceedings of the fifth international conference on Autonomous agents. AGENTS '01, New York, NY, USA, ACM (2001) 194–195
36. Ramchurn, S.D., Jennings, N.R., et al.: A computational trust model for multi-agent interactions based on confidence and reputation. In: IN PROCEEDINGS OF 6TH INTERNATIONAL WORKSHOP OF DECEPTION, FRAUD AND TRUST IN AGENT SOCIETIES. (2003) 69–75
37. Billhardt, H., Hermoso, R., Ossowski, S., Centeno, R.: Trust-based service provider selection in open environments. In: Proceedings of the 2007 ACM symposium on Applied computing. SAC '07, New York, NY, USA, ACM (2007) 1375–1380
38. Bertino, E., Ferrari, E., Squicciarini, A.: Trust negotiations: concepts, systems, and languages. Computing in Science and Engineering **6**(4) (July 2004) 27–34
39. Winslett, M., Yu, T., Seamons, K.E., Hess, A., Jacobson, J., Jarvis, R., Smith, B., Yu, L.: Negotiating trust on the web. IEEE Internet Computing **6** (November 2002) 30–37
40. Koshutanski, H., Massacci, F.: An access control framework for business processes for web services. In: Proceedings of the 2003 ACM workshop on XML security. XMLSEC '03, New York, NY, USA, ACM (2003) 15–24
41. Dragoni, N., Massacci, F.: Security-by-contract for web services. In: Proceedings of the 2007 ACM workshop on Secure web services. SWS '07, New York, NY, USA, ACM (2007) 90–98
42. Yu, T., Winslett, M., Seamons, K.E.: Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. ACM Trans. Inf. Syst. Secur. **6** (February 2003) 1–42
43. Ye, S., Makedon, F., Ford, J.: Collaborative automated trust negotiation in peer-to-peer systems. In: Proceedings of the Fourth International Conference on Peer-to-Peer Computing. P2P '04, Washington, DC, USA, IEEE Computer Society (2004) 108–115
44. Leithead, T., Nejdl, W., Olmedilla, D., Seamons, K.E., Winslett, M., Yu, T., Zhang, C.C.: How to exploit ontologies in trust negotiation. In: University of Aachen (RWTH. (2004) 127
45. Nejdl, W., Olmedilla, D., Winslett, M.: Peertrust: Automated trust negotiation for peers on the semantic web. In: In Workshop on Secure Data Management in a Connected World (SDM04. (2004) 118–132
46. Mecella, M., Ouzzani, M., Paci, F., Bertino, E.: Access control enforcement for conversation-based web services. In: Proceedings of the 15th international conference on World Wide Web. WWW '06, New York, NY, USA, ACM (2006) 257–266
47. Dragoni, N., Massacci, F., Saidane, A.: A self-protecting and self-healing framework for negotiating services and trust in autonomic communication systems. Comput. Netw. **53** (July 2009) 1628–1648
48. Rasmusson, L., Jansson, S.: Simulated social control for secure internet commerce. In: Proceedings of the 1996 workshop on New security paradigms. NSPW '96, New York, NY, USA, ACM (1996) 18–25

# Bibliography

[1] Liberty alliance. http://www.projectliberty.org/. url consulted on September 15, 2011.

[2] Cassandra: Distributed access control policies with tunable expressiveness. In *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 159–, Washington, DC, USA, 2004. IEEE Computer Society.

[3] Trust-x: A peer-to-peer framework for trust establishment. *IEEE Trans. on Knowl. and Data Eng.*, 16:827–842, July 2004.

[4] eXtensible Access Control Markup Language (XACML) Version 2.0. Technical report, OASIS Access Control TC, February 2005.

[5] Alfarez Abdul-Rahman and Stephen Hailes. Using recommendations for managing trust in distributed systems. In *IEEE Malaysia International Conference on Communication*, November 1997.

[6] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 17(6):734–749, 2005.

[7] Ali Shaikh Ali, Simone A. Ludwig, and Omer F. Rana. A cognitive trust-based approach for web service discovery and selection. Technical report, Department of Computer Science Cardiff University, UK, 2010.

[8] Ali Arsanjani. Service-oriented modeling and architecture. http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/, November 2004. url consulted on March 23, 2011.

[9] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Commun. ACM*, 40:66–72, March 1997.

[10] E. Bertino, E. Ferrari, and A. Squicciarini. X -tnl: An xml-based language for trust negotiations. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '03, pages 81–, Washington, DC, USA, 2003. IEEE Computer Society.

[11] E. Bertino, E. Ferrari, and A. Squicciarini. Trust negotiations: concepts, systems, and languages. *Computing in Science and Engineering*, 6(4):27–34, July 2004.

[12] Holger Billhardt, Ramón Hermoso, Sascha Ossowski, and Roberto Centeno. Trust-based service provider selection in open environments. In *Proceedings of the 2007 ACM symposium on Applied computing*, SAC '07, pages 1375–1380, New York, NY, USA, 2007. ACM.

[13] M. Bishop. *Computer security: art and science.* Addison-Wesley, 2003.

[14] Guido Boella, Università Di Torino, and Joris Hulstijn. Argument games for interactive access control. In *In Proc. of WI 2005*, pages 751–754, 2005.

[15] Piero A. Bonatti and Pierangela Samarati. A uniform framework for regulating service access and information release on the web. *J. Comput. Secur.*, 10:241–271, September 2002.

[16] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12:331–370, November 2002.

[17] Castelfranchi C. and Tan Y.H. *Trust and Deception in Virtual Societies.* Kluwer Academic Publishers, pres.

[18] Scott Cantor, John Kemp, Rob Philpott, and Eve Maler. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. Technical report, March 2005.

[19] C. Castelfranchi and R. Falcone. Principles of trust for mas: Cognitive anatomy, social importance, and quantification. In *Proceedings of the 3rd International Conference on Multi Agent Systems*, ICMAS '98, pages 72–, Washington, DC, USA, 1998. IEEE Computer Society.

[20] Elizabeth Chang, Farookh Hussain, and Tharam Dillon. *Trust and Reputation for Service-Oriented Environments: Technologies For Building Business Intelligence And Consumer Confidence.* John Wiley & Sons, 2005.

[21] Uddam Chukmol. A framework for web service discovery: service's reuse, quality, evolution and user's data handling. In *Proceedings of the 2nd SIGMOD PhD workshop on Innovative database research*, IDAR '08, pages 13–18, New York, NY, USA, 2008. ACM.

[22] John Domingue, Liliana Cabral, Stefania Galizia, Vlad Tanasescu, Alessio Gugliotta, Barry Norton, and Carlos Pedrinaci. Irs-iii: A broker-based approach to semantic web services. *Web Semant.*, 6:109–132, April 2008.

[23] Nicola Dragoni. A survey on trust-based web service provision approaches. Technical report, In Proc. of the 3rd International Conference on Dependability (DEPEND 2010), Venice, Italy, 2010, IEEE CPS.

[24] Nicola Dragoni. Turst-based service selection. University Lecture, 2010.

[25] Nicola Dragoni and Fabio Massacci. Security-by-contract for web services. In *Proceedings of the 2007 ACM workshop on Secure web services*, SWS '07, pages 90–98, New York, NY, USA, 2007. ACM.

[26] Nicola Dragoni, Fabio Massacci, and Ayda Saidane. A self-protecting and self-healing framework for negotiating services and trust in autonomic communication systems. *Comput. Netw.*, 53:1628–1648, July 2009.

[27] Thomas Erl. *SOA: Principles of Service Design.* Prentice Hall, first edition, 2007.

[28] Stefania Galizia, Alessio Gugliotta, and John Domingue. A trust based methodology for web service selection. In *Proceedings of the International Conference on Semantic Computing*, pages 193–200, Washington, DC, USA, 2007. IEEE Computer Society.

[29] W3C Group. Web service use case: Travel reservation. http://www.w3.org/2002/06/ws-example, May 2002. url consulted on March 25, 2011.

[30] W3C Group. Web services architecture. http://www.w3.org/TR/ws-arch//, February 2004. url consulted on March 24, 2011.

[31] W3C Group. Web services architecture usage scenarios. http://www.w3.org/TR/ws-arch-scenarios/, February 2004. url consulted on March 25, 2011.

[32] W3C Group. Web services choreography requirements. http://www.w3.org/TR/ws-chor-reqs/, March 2004. url consulted on March 26, 2011.

[33] W3C Group. Web services glossary. http://www.w3.org/TR/ws-gloss/, February 2004. url consulted on March 24, 2011.

[34] Robert H. Guttman, Alexandros G. Moukas, and Pattie Maes. Agent-mediated electronic commerce: a survey. *Knowl. Eng. Rev.*, 13:147–159, July 1998.

[35] Jim Heid. Business information in cyberspace. *Macworld*, 12(3):137–140, March 1995.

[36] Amir Herzberg, Yosi Mass, Joris Michaeli, Yiftach Ravid, and Dalit Naor. Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 2–, Washington, DC, USA, 2000. IEEE Computer Society.

[37] R. Housley, S. Spyru, W. Ford, Verisign, W. Polk, T. Nis, D. Solo, and Citicorp. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. January 1999.

[38] Michael N. Huhns and Munindar P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9:75–81, January 2005.

[39] Le hung Vu, Manfred Hauswirth, and Karl Aberer. Qos-based service selection and ranking with trust and reputation management. In *in Proceedings of the Cooperative Information System Conference (CoopISÕ05)*, pages 446–483, 2005.

[40] Dong Huynh, Nicholas R. Jennings, and Nigel R. Shadbolt. Developing an integrated trust and reputation model for open multi-agent systems. pages 65–74, 2004.

[41] Silicon Graphics International, editor. *IRIX Network Programming Guide 6.5*, chapter 4. 13th edition, July 2003.

[42] Alexander Linden Jackie Fenn. Gartner's hype cycle special report for 2005. *Gartner*, 2005.

[43] Catholijn M. Jonker and Jan Treur. Formal analysis of models for the dynamics of trust based on experiences. Technical report, Department of Computer Science Cardiff University, UK, 1999.

[44] Audun Jøsang. A logic for uncertain probabilities. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 9:279–311, June 2001.

[45] Audun Jøsang, Touhid Bhuiyan, Yue Xu, and Clive Cox. Combining trust and reputation management for web-based services. In *Proceedings of the 5th international conference on Trust, Privacy and Security in Digital Business*, TrustBus '08, pages 90–99, Berlin, Heidelberg, 2008. Springer-Verlag.

[46] Audun Jøsang and Jochen Haller. Dirichlet reputation systems. In *INTERNATIONAL CONFERENCE ON AVAILABILITY, RELIABILITY AND SECURITY*, pages 112–119. IEEE Computer Society, 2007.

[47] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decis. Support Syst.*, 43:618–644, March 2007.

[48] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 640–651, New York, NY, USA, 2003. ACM.

[49] Hristo Koshutanski and Fabio Massacci. An access control framework for business processes for web services. In *Proceedings of the 2003 ACM workshop on XML security*, XMLSEC '03, pages 15–24, New York, NY, USA, 2003. ACM.

[50] Hristo Koshutanski and Fabio Massacci. Interactive access control for web services. In *In Proceedings of the 19th IFIP International Information Security Conference (SEC 2004*, pages 151–166. Kluwer Press, 2004.

[51] Kyriakos Kritikos and Dimitris Plexousakis. Owl-q for semantic qos-based web service description and discovery. In Tommaso Di Noia, Rubèn Lara, Axel Polleres, Ioan Toma, Takahiro Kawamura, Matthias Klusch, Abraham Bernstein, Massimo Paolucci, Alain Leger, and David L. Martin, editors, *SMRR*, volume 243 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[52] Kyriakos Kritikos and Dimitris Plexousakis. Semantic qos-based web service discovery algorithms. *Web Services, European Conference on*, 0:181–190, 2007.

[53] Kyriakos Kritikos and Dimitris Plexousakis. Mixed-integer programming for qos-based web service matchmaking. *IEEE Transactions on Services Computing*, 2:122–139, 2009.

[54] Bruce Krulwich. Lifestyle finder: Intelligent user profiling using large-scale demographic data. *AI Magazine*, 18(2):37–45, 1997.

[55] Adam J. Lee, Marianne Winslett, and Kenneth J. Perano. Trustbuilder2: A reconfigurable framework for trust negotiation. In *IFIPTM*, pages 176–195, 2009.

[56] Travis Leithead, Wolfgang Nejdl, Daniel Olmedilla, Kent E. Seamons, Marianne Winslett, Ting Yu, and Charles C. Zhang. How to exploit ontologies in trust negotiation. In *University of Aachen (RWTH*, page 127, 2004.

[57] Jiangtao Li, Ninghui Li, and William H. Winsborough. Automated trust negotiation using cryptographic credentials. In *Proceedings of the 12th ACM conference on Computer and communications security*, CCS '05, pages 46–57, New York, NY, USA, 2005. ACM.

[58] Ninghui Li and John C. Mitchell. Rt: A role-based trust-management framework, 2003.

[59] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–, Washington, DC, USA, 2002. IEEE Computer Society.

[60] Pan Li, Meng Xiangxu, Shen Zhiqi, and Yu Han. A reputation pattern for service oriented computing. In *Proceedings of the 7th international conference on Information, communications and signal processing*, ICICS'09, pages 239–243, Piscataway, NJ, USA, 2009. IEEE Press.

[61] Kwei-Jay Lin, Jane Yung-jen Hsu, Yue Zhang, and Tao Yu. A distributed reputation broker framework for web service applications. *Journal of Electronic Commerce Research*, 7(3):164–177, November 2006. ABI/INFORM.

[62] Yuhong Liu and Yan (Lindsay) Sun. Anomaly detection in feedback-based reputation systems through temporal and correlation analysis. In *Proceedings of the 2010 IEEE Second International Conference on Social Computing*, SOCIALCOM '10, pages 65–72, Washington, DC, USA, 2010. IEEE Computer Society.

[63] Yan Lu, Zhaozi Gao, and Kai Chen. A dynamic composition algorithm of semantic web service based on qos. In *Proceedings of the 2010 Second International Conference on Future Networks*, ICFN '10, pages 354–356, Washington, DC, USA, 2010. IEEE Computer Society.

[64] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, Rebekah Metz, and Booz Allen Hamilton. *OASIS Reference Model for Service Oriented Architecture 1.0.* OASIS, October 2006.

[65] Umardand Shripad Manikrao. Dynamic selection of web services with recommendation system. In *In: Proceedings of the International Conference on Next Generation Web Services Practices (NWESP), IEEE Computer Society*, page 117. Press, 2005.

[66] Carolyn McLeod. Trust. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2011 edition, 2011.

[67] Massimo Mecella, Mourad Ouzzani, Federica Paci, and Elisa Bertino. Access control enforcement for conversation-based web services. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 257–266, New York, NY, USA, 2006. ACM.

[68] D. A. Menasce. QoS issues in Web services. *Internet Computing, IEEE*, 6(6):72–75, 2002.

[69] Wolfgang Nejdl, Daniel Olmedilla, and Marianne Winslett. Peertrust: Automated trust negotiation for peers on the semantic web. In *In Workshop on Secure Data Management in a Connected World (SDMÕ04*, pages 118–132, 2004.

[70] Daniel Olmedilla, Rubèn Lara, Axel Polleres, and Holger Lausen. Trust negotiation for semantic web services. In *1ST INTERNATIONAL WORKSHOP ON SEMANTIC WEB SERVICES AND WEB PROCESS COMPOSITION IN CONJUNCTION WITH THE 2004 IEEE INTERNATIONAL CONFERENCE ON WEB SERVICES*, pages 81–95. Springer, 2004.

[71] Michael J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artif. Intell. Rev.*, 13:393–408, December 1999.

[72] Falcone R., Pezzuolo G., and Castelfranchi C. *Special issue on 'Trust, Reputation and Security: Theories and Practice'*, chapter A - Fuzzy approach to a belief-based trust computation, pages 73–86. 2003.

[73] Sarvapali D. Ramchurn, Nicholas R. Jennings, and et al. A computational trust model for multi-agent interactions based on confidence and reputation. In *IN PROCEEDINGS OF 6TH INTERNATIONAL WORKSHOP OF DECEPTION, FRAUD AND TRUST IN AGENT SOCIETIES*, pages 69–75, 2003.

[74] Shuping Ran. A model for web services discovery with qos. *SIGecom Exch.*, 4:1–10, March 2003.

[75] Anand S. Rao and Michael P. Georgeff. Bdi agents: From theory to practice. In *IN PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON MULTI-AGENT SYSTEMS (ICMAS-95*, pages 312–319, 1995.

[76] Lars Rasmusson and Sverker Jansson. Simulated social control for secure internet commerce. In *Proceedings of the 1996 workshop on New security paradigms*, NSPW '96, pages 18–25, New York, NY, USA, 1996. ACM.

[77] Elaine Rich. Readings in intelligent user interfaces. chapter User modeling via stereotypes, pages 329–342. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[78] Alex Rodriguez. Restful web services: The basics. http://www.ibm.com/developerworks/webservices/library/ws-restful/, November 2008. url consulted on March 24, 2011.

[79] A. Ruiz-Cortés, O. Martín-Díaz, A. Durán-Toro, and M. Toro. Improving the Automatic Procurement of Web Services Using Constraint Programming. *Int. J. Cooperative Inf. Syst*, 14(4):439–468, 2005.

[80] Jordi Sabater and Carles Sierra. Regret: reputation in gregarious societies. In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, pages 194–195, New York, NY, USA, 2001. ACM.

[81] J. Ben Schafer, Joseph Konstan, and John Riedi. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, EC '99, pages 158–166, New York, NY, USA, 1999. ACM.

[82] M. Schillo, P. Funk, and M. Rovatsos. Who can you trust: Dealing with deception. In *Proceedings of Autonomous Agents '99 Workshop on "Deception, Fraud, and Trust in Agent Societies"*, pages 81–94, Seattle, USA, May 1999.

[83] S Schmitt and R Bergmann. Applying case-based reasoning technology for product selection and customization in electronic commerce environments. In *12th Bled Electronic Commerce Conference*, 1999.

[84] Munindar P. Singh, Bin Yu, and Mahadevan Venkatraman. Community-based service location. *Commun. ACM*, 44:49–54, April 2001.

[85] James Snell, Doug Tidwell, and Pavel Kulchenko. *Programming Web Services with SOAP*, chapter 3. O'Reilly Media, 1st edition, December 2001.

[86] A. Squicciarini, E. Bertino, Elena Ferrari, F. Paci, and B. Thuraisingham. Pp-trust-x: A system for privacy preserving trust negotiations. *ACM Trans. Inf. Syst. Secur.*, 10, July 2007.

[87] Anna Cinzia Squicciarini, Alberto Trombetta, and Elisa Bertino. Supporting robust and secure interactions in open domains through recovery of trust negotiations. In *Proceedings of the 27th International Conference on Distributed Computing Systems*, ICDCS '07, pages 57–, Washington, DC, USA, 2007. IEEE Computer Society.

[88] L. Taher and H. El Khatib. A framework and qos matchmaking algorithm for dynamic web services selection. In *In Proceedings of the 2 nd International Conference on Innovations in Information Technology (IIT$\tilde{\text{O}}$05*, 2005.

[89] M. Tian, A. Gramm, T. Naumowicz, H. Ritter, and J. S. Freie. A concept for QoS integration in Web services. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering Workshops (WISEW), Roma, Italy*, pages 149–155, December 2003.

[90] B. Towle and C. Quinn. Knowledge based recommender systems using explicit user models. In *Papers from the AAAI Workshop, AAAI Technical Report WS-00-04*, pages 74–77. Menlo Park, CA: AAAI Press, 2000.

[91] T.Rajendran, P.Balasubramanie, and Resmi Cherian. Article: An efficient ws-qos broker based architecture for web services selection. *International Journal of Computer Applications*, 1(9):79–84, February 2010. Published By Foundation of Computer Science.

[92] Sameer Tyagi. Restful web services. http://www.oracle.com/technetwork/articles/javase/index-137171.html, August 2006. url consulted on March 25, 2011.

[93] Yao Wang and Julita Vassileva. Toward trust and reputation based web service selection: A survey, 2007.

[94] Marianne Winslett, Ting Yu, Kent E. Seamons, Adam Hess, Jared Jacobson, Ryan Jarvis, Bryan Smith, and Lina Yu. Negotiating trust on the web. *IEEE Internet Computing*, 6:30–37, November 2002.

[95] Marianne Winslett, Charles C. Zhang, and Piero A. Bonatti. Peeraccess: a logic for distributed authorization. In *Proceedings of the 12th ACM conference on Computer and communications security*, CCS '05, pages 168–179, New York, NY, USA, 2005. ACM.

[96] Li Xiong and Ling Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 16:843–857, 2004.

[97] Ziqiang Xu, Patrick Martin, Wendy Powley, and Farhana Zulkernine. Reputation-enhanced qos-based web services discovery, 2007.

[98] Ziqiang Xu and Copyright Ziqiang Xu. Reputation-enhanced web service discovery with qos. In *Ph.D. Dissertation, School of Computing, QueenÕs*, 2006.

[99] Song Ye, Fillia Makedon, and James Ford. Collaborative automated trust negotiation in peer-to-peer systems. In *Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, P2P '04, pages 108–115, Washington, DC, USA, 2004. IEEE Computer Society.

[100] Pınar Yolum and Munindar P. Singh. Engineering self-organizing referral networks for trustworthy service selection. *IEEE Transactions on Systems, Man, and Cybernetics. Part A*, 35(3):396–407, 2005.

[101] Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Trans. Inf. Syst. Secur.*, 6:1–42, February 2003.

[102] Giorgos Zacharia, Alexandros Moukas, and Pattie Maes. Collaborative reputation mechanisms in electronic marketplaces. In *Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences-Volume 8 - Volume 8*, HICSS '99, pages 8026–, Washington, DC, USA, 1999. IEEE Computer Society.

[103] Jia Zhang, Liang-Jie Zhang, and Jen-Yao Chung. Ws-trustworthy: A framework for web services centered trustworthy computing. In *Proceedings of the 2004 IEEE International Conference on Services Computing*, pages 186–193, Washington, DC, USA, 2004. IEEE Computer Society.

[104] Bo Zhou, Keting Yin, Honghong Jiang, Shuai Zhang, and Aleksander Kavs. Qos-based selection of multi-granularity web services for the composition. *Journal of Software*, 6(3), 2011.