INCENTIVE-BASED REPUTATION OF WEB SERVICES COMMUNITIES

AHMAD MOAZIN

A THESIS

IN

THE CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF APPLIED SCIENCE

(INFORMATION SYSTEMS SECURITY) AT

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

JANUARY 2010

Your file Votre référence
ISBN: 978-0-494-67276-1
Our file Notre référence
ISBN: 978-0-494-67276-1

Abstract

Incentive-based Reputation of Web Services Communities

Ahmad Moazin

There have been always motivations to introduce clustering of entities with similar functionality into groups of redundant services or agents. Communities of Web services are composed by aggregating a number of functionally identical Web services. Many communities with the same type of service can be formed and all aim to increase their reputation level in order to obtain more requests. The problem, however, is that there are no incentives for these communities to act truthfully and not providing fake feedback in support of themselves or against others.

In this thesis we propose an incentive and game-theoretic-based mechanism dealing with reputation assessment for communities of Web services. The proposed reputation mechanism is based on after-service feedback provided by the users to a logging system. Given that the communities are free to decide about their actions, the proposed method defines the evaluation metrics involved in reputation assessment of the communities and supervises the logging system by means of controller agent in order to verify the validity and soundness of the feedback. We also define incentives so that the best game-theoretical strategy for communities is to act truthfully. Theoretical analysis of the framework along with empirical results are provided.

Acknowledgments

I would like to extend my sincerest thanks to my supervisor, Dr. Jamal Bentahar, for providing me with this opportunity. I would like to thank him for all his excellent guidance, precious advice and endless support over the years during my graduate study and research at Concordia University. I would also like to thank my colleague Babak Khosravifar for the priceless help, discussions, support, and experience we have shared.

I would like to thank my lab mates, fellow students and friends who have provided endless inspiration during my stay at Concordia University. Additionally, I would like to thank my relatives in Ottawa for their support.

Finally, I would like to express my sincerest appreciation and love to my parents for all their help and support to fulfill my dreams.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the context of research, which is about reputation of agent-based communities of Web services. It presents the motivations of this work and describes our contributions. The last section presents the thesis organization.

1.1 Context of Research

This thesis is about reputation of agent-based communities of Web services. Our main focus is on the reputation adjustment based on the after-service feedback submitted by the users of these communities. The reputation adjustment is reflected by a ranking list that classifies the communities according to the reputation value they hold at each time frame.

Web services are software systems that have recently been deployed to maintain inter-operable interactions between applications. Abstracting these Web services by software autonomous

agents will benefit them from flexible and intelligent interactions that agents are able to manage [10, 14, 15], and from the ability of overseeing their performances, commitments, and availability details. Agents are software systems that have the ability to react with the environment they reside in and learn from due to their autonomous behavior. Agent-based Web services are Web services having the capability to interact with their environment thanks to the agents that are assigned to serve as their representatives. However, because agents are autonomous and thus selfish [16], a challenging issue in the resulting environments of agent-based Web services is reputation, which is an important factor that regulates the process of selecting services. Reputation [37] in this context is a social evaluation of the users towards the communities of Web services where communities of high quality of service will normally have high reputation and thus more chance to be selected by prospective users.

On the other hand, overload and hence poor responsiveness are unavoidable issue that single Web services along with their associated agents should manage. Regarding this regulation, the concept of communities that gather agent-based Web services having the same functionality has been emerged, which provides a more reliable service response [5, 9]. Each community of Web services is led by a master Web service, which is responsible for accepting or inviting (hiring) new Web services to be members of the community and excluding or rejecting (firing) existing Web services. Deploying such a community is more cost-effective and thus, a proper management is required to handle community response. But still the issue of communities' reputation is yet to be addressed as these communities could follow strategies that lead to self support in the environment.

This thesis presents a reputation mechanism for communities of Web services that focuses on a feedback logging mechanism. Logging mechanism is a mechanism that supports logging

of the feedback from the users to a log file in the form of binary feedback (1 for positive and 0 for negative), where these feedback can then be extracted from the log file in the process of calculating community's reputation value. For the reputation mechanism, violations are detected by the means of a specific controller agent. Malicious communities are identified by this agent and penalties are applied. The controller agent Cg is an agent who is responsible of taking control over the logging mechanism and keeping the log file accurate.

The main objective of our work is to incentivize the communities of Web services in order to behave truthfully. This goal is achieved through the design of a suitable reputation mechanism to monitor and control the different behaviors of communities. The contributions of this thesis are discussed in more details in Section 1.3.

1.2 Motivations

In order to facilitate and enhance the selection process of Web services, communities of Web services are designed to effectively manage the requests from users. In the process of searching for and selecting a Web service, each user has to look for a community that offers his desired service where reputation is the main differentiation factor between different communities. The user can be a person searching for a service such as hotel reservation or software agent working on behalf of that user.

The reputation of communities of Web services has been studied in some other work [5, 9], which in turn motivate our research. The motivation is to find a way to keep communities' reputation ranking list an accurate source of information about the communities' quality in which the user can rely on without being misled by fake reputation values. In order to reach that objective,

we participate in proposing a reputation mechanism that helps in selecting a particular community and in providing after-service feedback regarding the received quality of service. Such mechanism allows the controller agent Cg to stay in track with the reputation level for each community during the lifetime of that community to make sure that communities are not trying to increase their reputation level values in a malicious way.

1.3 Thesis Contributions

This thesis is partially based on the work we did in [23] where we extended the work that has been done in [9] by two contributions. In the first contribution, we discuss an empirical reputation trust model for the communities of Web services, which is based on three involved metrics (*responsiveness, inDemand* and *satisfaction*). These factors are redefined in a different way by considering the time factor we call *time recency*. This model is used by users and providers of Web services to estimate the reputation of a community under consideration. The second contribution is about the logging mechanism, which is designed to be reliable (capable of managing malicious acts of communities and thus preserving the integrity of the logging file from unauthorized modifications).

In this model, we assume that communities may have strategies to violate the run-time logging mechanism in support of themselves or against other communities. To secure our reputation system's integrity from such an attack, we try to discover feedback violation using the controller agent Cg (the agent that is assigned to monitor and audit the logging data) that, to some extent, makes sure that the violation did happen. Then we propose a method to properly react by penalizing malicious communities and reward good ones.

We provide an analysis of the incentive for communities not to violate the logging system. The idea is to prove that communities gain more if they do not violate the logging system compared to when they violate it. In this analysis, we define the comparative values of rewards and penalties for communities in order to obtain such an incentive. The results of the simulations reveal how, empirically, our reputation model allows us to adjust the level of communities's reputation. What specifically distinguishes our model from other similar work in the literature [5, 9] is its incentive-based reputation adjustment in the sense that although the communities are capable of violating the integrity of the logging system in support of themselves (or against their opponents), they will not take the risk to do that, given the fact that they are aware of the auditing procedure by the controller agent Cg and the possible penalties, which can decrease their current reputation level.

In this thesis, we show that by using incentives and penalties, the best strategy for communities is to act truthfully. The advantages of using the incentive-based mechanism are:

- obtaining an accurate information for deriving the involved metrics used for the reputation of a particular community;
- obtaining an overall higher reliability in the sense that upon violation detection, communities are strictly encouraged to show an acceptable performance in their further user request processes.

Moreover, In this thesis, we report on a work in which we participate [23] and which advances the proposed framework by analyzing the system's involved parameters in a two-player game theoretic study. We investigate the incentives to cheat that malicious communities aim to take advantage of, and the incentives to act as normal while being aware of the possible penalties

assigned by the controller agent. In fact we empirically analyze the payoffs obtained following different strategies. In our experimental work we discuss the obtained results that enable us to elaborate more on the outcome of different strategies that players choose.

1.4 Thesis Overview

This chapter provides the motivating context for this work. The remainder of the thesis is organized as follows. Chapter 2 gives an overview of the related work in the area of communities of Web services and reputation. Chapter 3 describes the proposed incentive-based reputation model for Web services communities. It presents the evaluation of the proposed model. It also describes the simulation environment and discusses the results of the experiments. Chapter 4 presents the game-theoretic analysis along with simulation results. Chapter 5 concludes this work and points to directions for future research opportunities.

Chapter 2

Background and Related Work

In this chapter, we review the literature related to Web services, their communities and associated trust frameworks. In Section 2.1, the area of Web services is presented. The concept of communities of Web services is presented in Section 2.2. In Section 2.3, we focus on the reputation. Finally, Section 2.4 concludes the chapter.

2.1 Web Services

The term "Web services" has generated a debate over what is actually a Web service and what is not, and is often used in many different ways. For this thesis, we will use the definition of Web services used by the World Wide Web Consortium (W3C): "A Web service is a software system designed to support inter-operable machine-to-machine interaction over a network. It has an interface described in a machine-processable format specifically Web Service Definition Language (WSDL). Other systems interact with the Web service in a manner prescribed by its description using Simple Object Access Protocol (SOAP) messages, typically conveyed using

Hypertext Transfer Protocol (HTTP) with an Extensible Markup Language (XML) serialization in conjunction with other Web-related standards." [13].

The rest of this section will give an overview about what a Web service is, its architecture, operations and finally the different technologies the Web services need to function.

2.1.1 Basic Principals

A Web service has the following behavioral characteristics [6, 33]:

1. **Loose coupling:** the interdependency between an application and a Web service should be as minimum as possible. For instance, if one of them changes that should not disturb the working of the other.

2. **Coarse grained approach:** the Web services technology provides an interface at a very high level of abstraction, allowing the developers to connect their applications to the desired Web services with minimum interactions.

3. **Synchronous and asynchronous mode of communication:** Web services support both synchronous and asynchronous mode of communication. But because Web services interact with so many components at the same time, asynchronous mode of communication is commonly used.

2.1.2 Architecture

The basic Web Services architecture defines an interaction between software agents as an exchange of messages among service requesters and service providers. Requesters are users software agents that request the execution of services. Providers are agents that provide services.

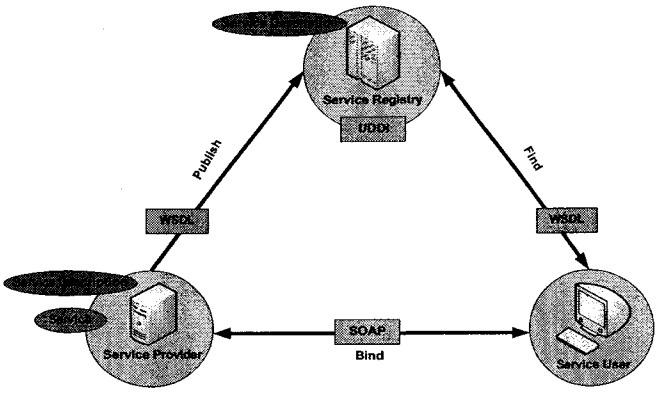


Figure 2.1: Web services architecture

Providers are responsible for publishing a description of the service they provide. Requesters must be able to find the description of the service. The Web Services architecture is based on the interactions between three roles [13]: *service provider*, *service registry* and *service requester* (see Figure 2.1).

1. **Service provider:** characterized by the platform that hosts access to the service.

2. **Service requester:** characterized by the application that is looking for and invoking or initiating an interaction with a service.

3. **Service registry:** is where service providers publish their service descriptions. Service requesters find services and obtain binding information.

2.1.3 Operations

For an application to take advantage of Web services, three behaviors must take place: *publishing* of service descriptions, *finding* or lookup of service descriptions, and *binding* or invoking of services based on the service description [2]. In detail, these operations are:

1. **Publish:** a service description needs to be published so that the service requester can find it. Where it is published can vary depending upon the requirements of the application.

2. **Find:** the service requester retrieves a service description directly or queries the service registry for the type of service required.

3. **Bind:** a service needs to be invoked. In the bind operation, the service requester invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact and invoke the service.

2.1.4 Standards Stack

Standards stack of Web services is composed by:

UDDI: The Universal Description, Discovery, and Integration Protocol

The Universal Description, Discovery, and Integration (UDDI) protocol is one of the Web services protocols that constitutes the Web services standards stack. This protocol provides an open and independent platform that enables users to find and locate service providers by searching UDDI registry for data and metadata representation about Web services [11,29].

XML: The Extensible Markup Language

The basic structure of XML is the document written in a description language, which is accepted as format to make the exchange of data possible. This terminology, however, might cause one to think of XML as only a richer, more flexible than HTML. Indeed, it can be so much more as well by allowing one to represent the data in a standard and structured format [29,42].

WSDL: The Web Services Description Language

Web Services Description Language (WSDL) is a format for describing a Web services interface. It is a way to describe services and how they should be bound to specific network addresses. This document must be created by the provider, and is exchanged between the provider and user through the service registry [29,40].

SOAP: The Simple Object Access Protocol

SOAP provides the envelope for sending Web services messages over the Internet given that Web services run in a heterogenous and distributed environment, which makes it necessary to use a independent protocol to carry out the data transfer between the different parts. SOAP commonly uses HTTP, but other protocols such as Simple Mail Transfer Protocol (SMTP) may be used. SOAP can be used to exchange complete documents or to call a remote procedures [29,41].

2.2 Communities of Web Services

Due to the fact that the number of Web services offering the same functionality continues to increase, the competition between them to attract new users is increasing as well. This fact of having a high number of Web services makes it harder for the user to select one service, as he is faced with many possibilities corresponding to his needs. Aggregating Web services with similar functionality into groups, or communities, can save the prospective user a lot of time and effort in the process of service discovery and selection. [4, 5, 39]. A community is considered as "a means to provide the description of a desired functionality without explicitly referring to any concrete Web service that will implement this functionality at run-time" [5]. The use of the

communities makes it possible to accelerate the discovery process based on the users' needs and improve the Web services reliability. This is because if one Web service cannot handle the user's request, another Web service from the same community can still handle it. But still, service users and providers must have the incentives to use the communities of Web services. From the user's perspective, it requires that the quality of service of the community must be greater than the quality of service (QoS) of the individual Web services. While from the provider's point of view, joining a community must have more value than being alone. Thus, it is essential that its participation rate alone is lower than that inside the community.

2.2.1 Definitions and Architecture

The architecture of Web services is shown in figure 2.2 (from [5]). In this figure we can see the different components that constitute that architecture: communities, providers of Web services, masters Web services, members Web services and UDDI registries. Each of these components has a role to play in the architecture as we will explain below.

When a Web services provider develops a new Web service, he advertises his new service in the UDDI registries by publishing its description such as its functionality, quality of service (QoS), reliability and so on. The master has a special role with some responsibilities within the community like the attraction of new Web services or selection of the Web services to meet the users' requests. The other Web services are considered as members and have in common the functionality of the community to which they belong. The master consult the UDDI registries for Web services that has the same functionality as his community. Each community is composed of a master of Web services and some other members Web services (Figure 2.2). The master has to

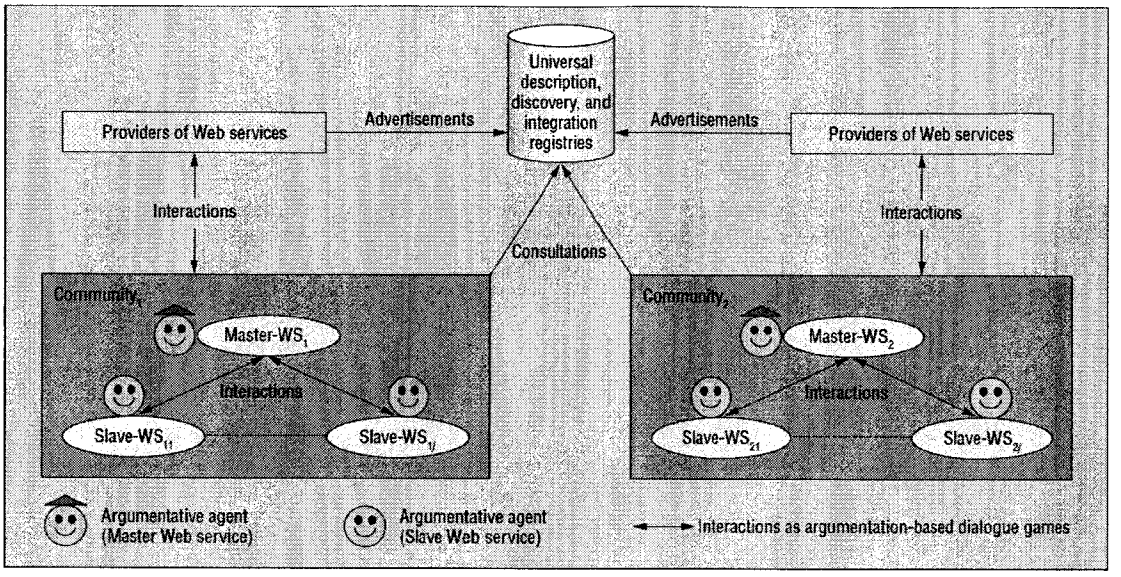


Figure 2.2: Web services, communities, and argumentative agents.

lead and manage the community and the members. Although the members within a community offer the same functionality, they have some different non-functional properties that make it possible to differentiate them. Nevertheless, there is some competition between them to be selected to satisfy the users' requests. A community is dynamic, which means some new Web services can join the community, whereas others leave it, some become unavailable and others are back after a suspension and so on. The master must be aware of these events in order to avoid some conflicts and to better monitor its community.

2.2.2 Community Development and Dismantlement

To develop a community of Web services, the designer has to set the the type of the community he is creating. The type of the community is also called the functionality such as Hotel reservation. Choosing the functionality of the community is the first step that the designer or the developer has to do in order to set up a community. The second step is to find a Web service that acts as

master of the community. There are two ways to choose a master for the community. In the first way, the designer designates a dedicated Web service to be the master throughout the life of the community. This Web service will never satisfies any request of the users. Its role is only to manage the community. In the second way, one member Web service is to be selected to act as master of the community. In this case, the master can be designated on a voluntary basis or after an election between the Web services.

Dismantling a community is also determined by the designer. When he creates it, he sets up some threshold, that must be respected by the members. However, "If the number of Web services in the community is less than a threshold and if the number of participation requests in composite Web services that arrive from users over a certain period of time is less than another threshold, then the community will be dismantled" [5]. In the case where a community does not respect the thresholds established by its designer, it is to be deleted and the Web services populating that community has to either work alone or to find another community that matches their functionality.

2.2.3 Web Services Attraction and Retention

Web service providers always publish or advertise their newly developed services to the UDDI registries. The master Web service, who is taking responsibility of managing the community he belongs to, interacts with the UDDI registries in a regular manner to stay up-to-date with any new advertisements about Web services that he may asks to be members of his community [4, 5].

The master Web service then, has to attract new Web services to be part of his community. The attraction process can be done in two common ways. The first way is by checking the

UDDI registries as mentioned above, by doing that the master will be searching for Web services that matches his communities' functional requirement or properties. When trying to attract a potential Web service, the master uses some arguments that help him to convince the potential Web service about the joining benefits. Some of the arguments that the master can use are: the high participation of its members in serving users, the security within the community, and the good reputation of the community that in turn bring more users. The second way to attract new Web service occurs when that potential Web service approaches the master and showing interest in being a member of his community. In this second way, the Web service has to use some arguments to convince the about the benefits that it will bring to his community. The master Web service has then to check its QoS, its reliability and capacity among many other properties. In case that a Web service matches the master's expectation, it will be registered in his community and become a member that can handle requests redirected to it by the master [4, 5].

A community with too few members will be deleted. So, it is important to keep as many existing members as possible. The fact that Web services stay in the same community for a long period means that the Web services are somehow satisfied with the participation rate about the users' requests. The master may also decide to fire a Web service if the Web service may have a new functionality that does not correspond to that of the community. In this case, the Web service is invited to join another community whose functionality is better suited. Another possibility is that the Web service becomes unreliable and cause troubles. For instance the master realizes that the Web service has cheated about its credentials in order to be accepted or to be selected in more composition scenarios and now it decreases the level of the community. It is also possible that a member decides to leave the community because of the low requests to that community in general [4, 5].

2.3 Reputation Systems

Reputation mechanism is a way to foster trust in online interactions by allowing members of a community to submit their opinions regarding other members of that community [37]. Submitted feedback are analyzed, aggregated with feedback posted by other members and made publicly available to the community in the form of member feedback profiles. Reputation is a variable, which depends on feedback about an interacting party's past behavior given by others and it affects the interacting party's future payoffs [8].

2.3.1 Reputation of Web Services

In the literature, the reputation of Web services has been intensively stressed [17,18,20,26,30,31, 38,45] aiming to facilitate and automate the good service selection. In [1], the authors introduce a framework aiming to select Web services based on the trust policies expressed by the users. The framework allows the users to select a Web service matching their needs and expectations. In [44], the authors propose an indirect trust mechanism aiming at establishing trust relationships from extant trust relationships with privacy protection. In [27], the authors propose to compute the reputation of a Web service according to the personal evaluation of the previous users. In the aforementioned models, the reputation of the Web service is measured by a combination of data collected from users. To this end, the credibility of the user that provides this data should be taken into account. There should be a mechanism that recognizes the biased rates provided from the users and accordingly updates their credibility. If the user tries to provide a fake rating, then its credibility will be decreased and the rating of this user will have less importance in the reputation of the Web service. In [32], the authors design a multi-agent framework based on an

ontology for QoS. The users' ratings according to the different qualities are used to compute the reputation of the Web service.

In general, in all the mentioned models, Web services are considered to act individually and not in collaboration with other services. In such systems, the service selection process is very complicated due to their relatively high number in the network. In addition, Web services can easily rig the system by leaving and joining the network when they benefit more from doing this. This is a rational incentive for such Web services that manage to start as new once they have shown a low efficiency.

2.3.2 Reputation of Communities

Regarding the aforementioned issue, there have been some proposals that try to gather Web services and propose the concept of community-based multi-agent systems [9, 12, 21]. In [9], the authors propose a reputation-based architecture for communities and classify the involved metrics that affect the reputation of a community. They define some metrics by processing some historical performance data recorded in a run-time logging system. The purpose is to be able to analyze the reputation in different points of view, such as users to communities, communities to Web services, and Web services to communities. The authors discuss the effect of different factors while diverse reputation directions are analyzed. However, they do not define the overall reputation of a community from the proposed metrics. Failing to assess the general reputation for the community leads to failure in efficient service selection. Moreover, authors assume that the run-time logging mechanism is an accurate source of information, which is not the case in real applications.

In general, in open reputation feedback mechanisms, always the feedback file is subject to be the target by malicious agents. The feedback mechanism should be supervised and its precise assessment should be guaranteed. In [21], the authors develop a framework that explores the possibilities that the active communities act truthfully and provide their actual information upon request. This method is related to the ideas that will be discussed in this thesis in the sense that the communities have incentives promoting them to act truthfully. The key idea behind their work is to promote truthfulness among agents and communities with the ultimate goal of increasing the social welfare of each community participating in the reputation mechanism. The assumption made in this work is that communities are self-interested and have the incentives to misreport the evaluation of their agents, and to address this problem of selfishness and fake reporting, they introduce a graph-based heuristic which gives the communities the incentive to provide truthful reports. The incentive they propose is a reward based on a payment mechanism [22], which pays honest communities for truthful reporting. In [12], a layered reputation assessment system is proposed mainly addressing the issue of anonymity. The focus is on the layered policies that are applied to measure the reputation of different types of agents, specially the new comers. The layered reputation mechanism classifies participants of peer-to-peer environments into layers or communities according to their reputation values. Participants in higher communities are those with high reputation while participants in the bottom layer or community have a low reputation. They key idea is the reputation adjustment, where bad behavior of participants from high communities will be punished and cause a severe drop in their reputation values. Although, the proposed work is interesting in terms of anonymous reputation assessment, the layered structure does not optimally organize a community-based environment that gathers equal Web services and also the computational expenses seems to be relatively high.

2.4 Conclusion

To address the aforementioned problems, we elaborate in this thesis on the reputation mechanism that is supervised by the controller agent Cg (an agent who is responsible of taking control over the logging mechanism and keeping the log files accurate) and based on the incentives provided to encourage more truthful actions. What mainly distinguishes our proposed model from the related work in the literature is its detailed focus on the logging mechanism accuracy and reputation assessment. The reputation system is observed by the controller agent but still communities are free in choosing their strategies. The incentive-based system provides a mechanism that guarantees the least fake actions since communities that gain benefit from malicious acts are eventually penalized so that their further decisions are altered.

Chapter 3

Reputation of Web Services Communities with Incentives

In this chapter, we define the architecture of reputation-embedded communities of Web services in Section 3.1. In Section 3.2, we discuss the reputation model by its involved metrics and a methodology to combine them. In Section 3.3, we extend the discussion about maintaining a feedback logging mechanism and we discuss the fake positive and negative corrections. In Section 3.4, we discuss the theoretical analysis for reputation level updates. In Section 3.5, we discuss the simulation. Finally, Section 3.6 concludes the chapter.

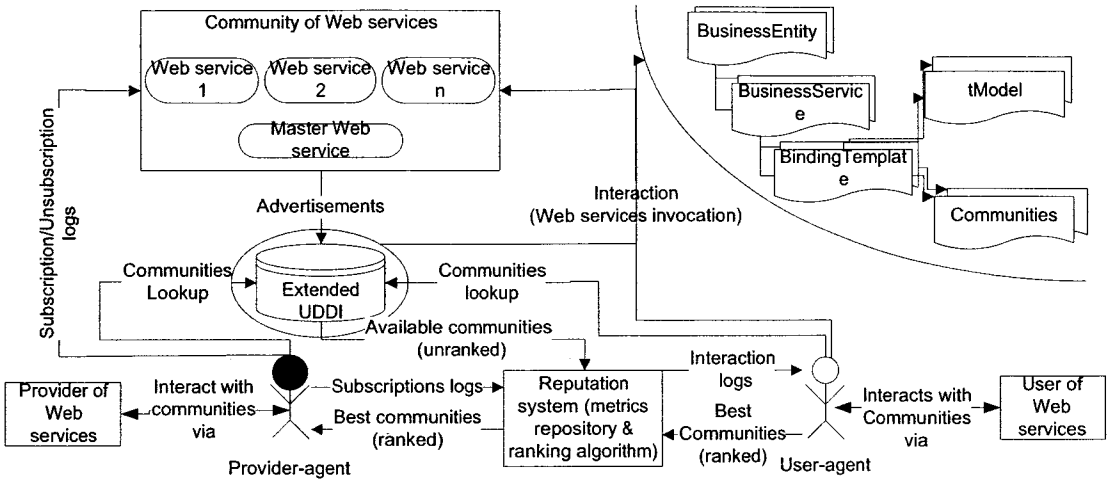


Figure 3.1: Architecture of reputation-based communities of Web services

3.1 Architecture of Reputation-Embedded Web services Communities

In this section, we present the communities of Web services architecture [9], which is designed to maintain the reputation of the communities of Web services. Figure 3.1 illustrates different components of the architecture and are as follows:

Extended UDDI: the traditional UDDI [36] is a registry that keeps the information about how to invoke all the registered Web services. While in the extended UDDI registry, the agents have a restricted access in the sense that user agents only consult the list of masters, whereas the masters have access to the list of the Web services in the UDDI registry. By adding this new information concerning the communities, we would identify which community a Web service belongs to.

User agent: is the client application program that is used as a representative of the user. This agent interacts with other parties such as the *extended UDDI*, communities of Web services and the reputation system.

Master Web service: is the agent that is representing and managing the community. This agent interacts with the users agents and handles their requests. It is also responsible for selecting a Web services from its community to serve the users. The master agent hires (or fires) some Web services to join (or leave) the community. In general, the master of the community always tends to increase the community's performance and its reputation level.

Provider agent: is the provider application program that is used as a representative of the provider. It interacts with other parties such as the *extended UDDI*, communities of Web services and the reputation system.

Reputation system: communities are always competing in order to obtain more requests. Their evaluations are then quite useful for users and providers. In order to assess the reputation of these communities, the user and provider agents must gather operational data about the communities performance. The reputation system is the core component in this architecture. Its main functionality is to rank the communities based on their reputation by using a ranking algorithm that makes use of run-time logs. The ranking algorithm would maintain a restrictive policy, avoiding the ranking violation, which could be done by some malicious communities. The violation could be done by providing some fake logging data (by some colluding users) that reflect positive feedback in support of the community, or by fake negative data that is registered against a particular community. To deal with this violation, we propose to assign a controller agent Cg.

Controller agent: Cg is the assigned agent that takes the logging file under surveillance and updates the assigned reputations to the communities. Cg is mainly responsible of removing the cheated feedback that support particular communities. By investigating the recent feedback, Cg recognizes the fake feedback and accordingly analyzes the further actions of the community. In general, Cg may fail to accurately detect the fake feedback or similarly may recognize normal

feedback as fake. In their strategies, malicious communities always consider this fake detection and analyze their chance of successful cheating.

3.2 Reputation System

The main issue of the reputation system is to use the collected reputation metrics to assess the total reputation of communities. In order to assess this reputation, the user needs to take some factors into account. In Subsection 3.2.1, we present the involved metrics that a user may consider in this assessment. Consequently, in Subection 3.2.2, we explain the methodology we use to combine these metrics in order to assess the reputation of a community.

3.2.1 Reputation Assessment Metrics

The reputation of a community is assessed according to the following metrics:

Responsiveness Metric: Let C be the community that is under consideration by user U. Responsiveness metric depicts the time to be served by a community. Let Res_C^{U,R^t} be the time taken by the master of the community C to answer the request received at time R^t by the user U. This time includes the time for selecting a Web service from the community and the time taken by that Web service to provide the service for the user U. For simplicity reason, we will not use the community C in the notation when it is clear from the context, for example the responsiveness metric will be denoted Res^{U,R^t}. Equation 1 [23] computes the response time of the community C, computed with U during the period of time $[t_1, t_2]$ ($Res^{U,[t_1,t_2]}$), where n is the number of

requests received by this community from U during this period of time.

$$Res^{U,[t_1,t_2]} = \frac{1}{n}\sum_{t=t_1}^{t_2} Res^{U,R^t} \times e^{-\lambda(t_2-t)} \tag{1}$$

Here the factor $e^{-\lambda(t_2-t)}$, where $\lambda \in [0,1]$, reflects the *time recency* of the received requests so that we can give more emphasize to the recent requests. If no request is received at a given time t, we suppose $Res^{U,R^t} = 0$.

InDemand Metric: It depicts the users' interest for a community C in comparison to the other communities. This factor is computed in equation 2 [23].

$$InD^{[t_1,t_2]} = \frac{Req^{[t_1,t_2]}}{\sum_{k=1}^{M} Req_{C_k}^{[t_1,t_2]}} \tag{2}$$

In this equation, $Req^{[t_1,t_2]}$ is defined as the number of requests that C has received during $[t_1,t_2]$, and M represents the number of communities under consideration.

Satisfaction Metric: Let Sat^{U,R^t} be a feedback rating value representing the satisfaction of user U with the service regarding his request R^t sent at time t to C. Equation 3 [23] shows the overall satisfaction during the time interval $[t_1,t_2]$.

$$Sat^{U,[t_1,t_2]} = \frac{1}{n}\sum_{t=t_1}^{t_2} Sat^{U,R^t} \times e^{-\lambda(t_2-t)} \tag{3}$$

3.2.2 Metrics Combination

In order to compute the reputation value of a community (which is between 0 and 1), it is needed to combine these metrics in a particular way. Actually, the *Responsiveness* and *Satisfaction*

metrics are the direct evaluations of the interactions between a user and a community whereas the *inDemand* metric is an assessment of a community in relation to other communities. In the first part, each user adds up his ratings of the *Responsiveness* and *Satisfaction* metrics for each interaction he has had with the community. Equation 4 [23] computes the reputation of the community C during the interval $[t_1, t_2]$ from the user U's point of view. In this equation, ν represents the maximum possible response time, so that if a community does not respond, we would have $Res^{U,[t_1,t_2]} = \nu$. In the second part, the *inDemand* metric is added. Therefore, the reputation of C from the users' point of view is obtained in equation 5 [23].

$$Rep^{U,[t_1,t_2]} = \eta(1 - \frac{Res^{U,[t_1,t_2]}}{\nu}) + \kappa Sat^{U,[t_1,t_2]} \tag{4}$$

$$Rep^{[t_1,t_2]} = \chi \frac{1}{m} \sum_{j=1}^{m} \left(Rep^{U_j,[t_1,t_2]}\right) + \phi \; InD^{[t_1,t_2]} \tag{5}$$

Where $\eta + \kappa = 1$ and $\chi + \phi = 1$.

The coefficients η, κ, χ, ϕ are generic values and all are related to Cg's control over the environment.

3.3 Feedback Logging Mechanism

In an environment composed of communities of Web services, master agents are selfish and may alter their intentions in order to obtain more popularity among users. This could happen by maliciously improving one's reputation level or by degrading other's reputation level or both. We respectively refer to these cases as fake positive/negative alteration. Therefore, it is important to avoid such attacks and keep the logging mechanism accurate [19]. In the rest of this section,

we focus on how to perform fake positive corrections and thus effectively maintain a reputation adjustment.

In the proposed architecture for the communities of Web services (Section 3.1), the reputation is computed based on the information obtained from the logging system used by the users during the elapsing time to leave their feedback (Section 3.2). It is the responsibility of the controller agent Cg to maintain an accurate attack-resilient logging file. In the proposed reputation system, Cg updates the total reputation of communities based on its surveillance over the provided feedback. Cg basically penalizes malicious communities for the faked feedback that they managed to provide via collusion with some users. Hereby, Cg also provides incentives to discourage malicious acts. We assume that this agent is highly secured in order to avoid being compromised.

Two cases of fake positives and fake negatives can be identified, but in this thesis we only focus on the fake positive case in the next subsection. In the surveillance system, there are three time spots t_0, t_1, and t_2 at which the reputation is measured and updated according to Cg's decisions over incidents.

3.3.1 Fake Positive Correction

In this subsection we will discuss the fake positive correction case.

Fake positive recognition. One of the main responsibilities of the controller agent Cg is to perform fake positive correction. Initially, Cg should recognize a malicious behavior from one or a set of user agents (that could possibly collude with a particular community). This recognition is done based on the recent observable change in the reputation of a community. To perform the recognition efficiently, Cg would always check the feedback of the communities.

So, Cg would consider the reputation that is computed for a specific period of time $[t_1 - \epsilon, t_1]$, where t_1 is the current time. The value ϵ measures the width of the time window the controller agent considers to check the recent feedback (ϵ as relatively small). Otherwise, Cg would take even older feedback into account (ϵ as relatively large). Thus, $Rep^{[t_1-\epsilon,t_1]}$ is the reputation of the community C obtained from data measured from $t_1 - \epsilon$ to t_1.

Let $U^{[t_1-\epsilon,t_1]}$ be the set of users that during this time interval provided a feedback for the community C, and t_b ($t_b < t_1 - \epsilon$) be the beginning time of collecting feedback. Cg would consider the positive feedback to be suspicious if the reputation improvement ($Rep^{[t_1-\epsilon,t_1]} - Rep^{[t_b,t_1]}$) divided by the number of users that caused such improvement is greater than a predefined threshold ϑ, i.e:

$$\frac{Rep^{[t_1-\epsilon,t_1]} - Rep^{[t_b,t_1]}}{|U^{[t_1-\epsilon,t_1]}|} > \vartheta$$

ϑ is set in the simulations taking into consideration that communities cannot manage more than a maximum number of users by time unit considering their sizes (i.e. the number of Web services populating the communities).

Fake positive Adjustment. Exceeding the threshold ϑ means that a particular community is probably receiving consequent positives. Then Cg would freeze the recent positive logs and notifies the corresponding community of this action. The idea is to observe the upcoming behavior (in terms of satisfaction and responsiveness) of the community in order to compare the actual efficiency with the suspended enhanced reputation level. During this period, the community has incentive to behave in such a way that reflects the suspended enhanced reputation level. As it is shown in Figure 3.2, the community's feedback is recognized as suspicious at time t_1. feedback

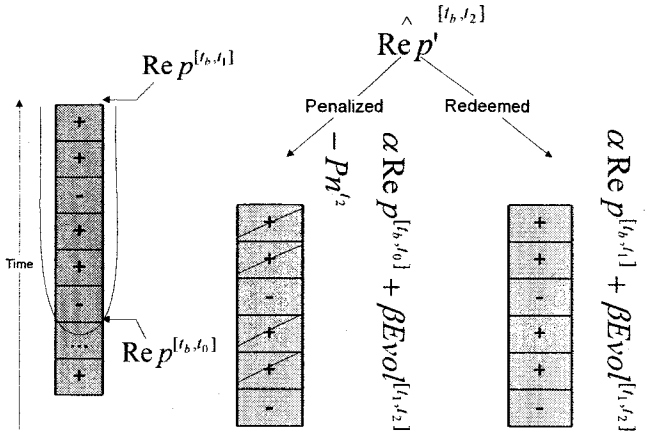


Figure 3.2: Fake positive correction cases

from time t_0 ($t_0 = t_1 - \epsilon$) are frozen to investigate the further behavior of the suspicious community C. At time t_2, controller agent Cg would decide whether to penalize community C or to redeem the frozen feedback. If the community has exhibited the real improved performance, the suspended reputation trust level would be considered for his reputation. However if the community has failed to do so, it will be penalized by decreasing its reputation. In this case, the community would be in such a situation that either has to outperform its past in order to improve the enhanced reputation level, or would loose its current reputation, which is not wanted.

This is then an incentive for communities to not jeopardize their current reputation level and thus not collude with users or providers by providing fake positives in support of themselves.

Let $Evol^{[t_1,t_2]}$ be the evolutionary reputation value for the community C that is measured by the controller agent Cg during specified time interval $[t_1, t_2]$ (investigation period). This value is computed in equation 6, where δ is a small value that the reputation is measurable within $[t-\delta, t]$.

$$Evol^{[t_1,t_2]} = \frac{\sum_{t=t_1+\delta}^{t_2} Rep^{[t-\delta,t]}}{t_2 - t_1} \tag{6}$$

Also, let P^t be the general penalty value that is assigned by Cg at a specific time t. Equation 7 computes the adjusted reputation level of C ($\widehat{Rep}^{[t_b,t_2]}$). This equation reflects the incentive that we propose, so that communities in general would be able to analyze their further reputation adjustments upon fake action.

$$\widehat{Rep}^{[t_b,t_2]} = \begin{cases} \alpha Rep^{[t_b,t_1]} + \beta Evol^{[t_1,t_2]}, & \text{if redeemed;} \\ \alpha Rep^{[t_b,t_0]} + \beta Evol^{[t_1,t_2]} - P^{t_2} & \text{if penalyzed.} \end{cases} \tag{7}$$

where $\alpha + \beta = 1$.

As discussed before, Cg will decide to redeem the community C if the evolutionary value for the reputation is more than C's previous reputation value, i.e.: $Evol^{[t_1,t_2]} \geq Rep^{[t_b,t_0]}$. If Cg decides to redeem the community C, then the previous reputation value (from time t_b to investigation time at t_1) is added to the evolutionary reputation value as a result of investigation during $[t_1, t_2]$. If Cg decides to penalize the community C, then the previous reputation is considered regardless of the improved reputation obtained in the period of $[t_0, t_1]$, and in addition to the evolutionary reputation, a penalty P^{t_2} is applied at time t_2.

3.4 Theoretical Analysis of the Reputation

In this section, we would like to discuss in details the updates of reputation level when a particular community C causes fake feedback that is eventually beneficiary for itself. To this end, we follow the steps over this reputation updates and elaborate Cg's actions on them.

To better analyze the decisions the communities could make, we calculate the expected reputation value of a particular community C in the case that the community acts maliciously to

provide fake positive feedback for itself and the case that the community acts as normal and performs its actual capabilities. By comparing the two expected values, the typical community C will decide either to act maliciously or as normal.

As discussed earlier, the decision to act maliciously or normal is made based on the probability to have a successful act, which is estimated by the community C. Being malicious, C always looks for cheating opportunities to increase its current reputation.

Let q^t be the probability that the controller agent Cg notices the real intention of the community C and take actions with penalizing C at time t. We compute the expected reputation of C as a result of a malicious action in equation 8 and as a result of normal action in equation 9. In these equations, the expected value of the reputation for community C is measured under two assumptions. In the case that C has faked the feedback ($E(\widehat{Rep}^{[t_b,t_2]}|C\ faked)$), the community decides to fake at time t_0 (therefore, the reputation till t_0 is considered as normal), the biased feedback are recognized by Cg at time t_1, and the investigation is finalized at time t_2. To this end, by penalizing C, its previous reputation till t_0 is considered together with the investigation period $[t_1, t_2]$ with its penalty. If the controller agent Cg does not recognize C's malicious act, all the feedback are taken into account. In this analysis, we consider a very low possibility that Cg warns false negatives, which is the case that Cg falsely recognizes a malicious act. To this end, we assume that if the community C acts as normal, the reputation value would be measured as normal.

$$\begin{aligned} E(\widehat{Rep}^{[t_b,t_2]}|C\ faked) = \\ q^{t_2}(\alpha Rep^{[t_b,t_0]} + \beta Evol^{[t_1,t_2]} - P^{t_2}) \\ + (1 - q^{t_2})(\alpha Rep^{[t_b,t_1]} + \beta Evol^{[t_1,t_2]}) \end{aligned} \tag{8}$$

$$E(\widehat{Rep}^{[t_b,t_2]}|C\ not faked) = Rep^{[t_b,t_2]} \tag{9}$$

As it is illustrated, the community that provides fake positives, obtains an improvement, which could be followed by a penalty. An important issue is that the probability of Cg's detection given the fact that the community C has faked before is high. Therefore, if C has been already penalized, it is so hard to retaliate and improve again. There is a slight chance that C fakes again and Cg ignores, which comes with a very small probability. Thus, we compute the expected reputation level of both cases and compare them.

Definition 3.1. *Let $Imp^{[t_b,t_2]}$ be the difference between the adjusted reputation (in the case where the community is under investigation) and normal reputation (in the opposite case) within $[t_b, t_2]$, i.e:*

$$Imp^{[t_b,t_2]} = \begin{cases} \widehat{Rep}^{[t_b,t_2]} - Rep^{[t_b,t_0]}, & \textit{investigated by } Cg; \\ Rep^{[t_b,t_2]} - Rep^{[t_b,t_0]}, & \textit{otherwise.} \end{cases}$$

The following proposition gives the condition for the penalty to be used, so that the communities will not act maliciously.

Proposition 3.1. *If $P^{t_2} > \frac{1}{q^{t_2}} Imp^{[t_b,t_2]} - \alpha Rep^{[t_0,t_1]}$, then communities obtain less reputation value if they act maliciously and provide fake positive feedback for themselves.*

Proof. To prove the proposition, we should consider the condition true and prove that $E(\widehat{Rep}^{[t_b,t_2]}|C\ faked) < E(\widehat{Rep}^{[t_b,t_2]}|C\ Not\ faked)$. By simple calculation we get:

$$E(\widehat{Rep}^{[t_b,t_2]}|C\ Not\ faked) - E(\widehat{Rep}^{[t_b,t_2]}|C\ faked) =$$
$$P^{t_2} - \frac{1}{q^{t_2}} Imp^{[t_b,t_2]} + \alpha Rep^{[t_0,t_1]}$$

The obtained value is positive, so it is done. □

3.5 Experimental results

In this section we describe the simulation experiments to illustrate the effectiveness of the reputation model. We first discuss the simulation environment we use to carry our experiments, and then we analyze the efficiency of the sound logging mechanism.

3.5.1 Simulation Environment

In our simulated system, communities host different agent-based Web services that are implemented as $Java^{©TM}$ agents. Agents deploy reasoning modules used to make decisions and select strategies. Indeed, time and environment features have impact on different agents' reasoning processes. The simulated system also hosts a number of users simulated by $Java^{©TM}$ agents responsible to search for services. Users initiate interactions with communities and provide after-interaction feedback reflecting quality of the received services. These users could collude with some communities if being encouraged to.

In order to implement the simulation environment, we have used NetBeans IDE 6.5 [7] as our Integrated Development Environment. It is an open-source Integrated Development Environment (IDE) for Java programming language.

Simulation Basics

In our implemented system, communities, their masters and Web services members are simulated.

Communities: the system starts by the simulation and creation of the communities. Each community has a unique ID, and will gather Web services that offers similar functionality, for example "Airline reservation service" in our case. The community quality of service (QoS) is the

average of the Web services' QoS.

Masters: the second step is the creation of master Web services. Each master has a unique ID and is associated with the ID of the community that this master is responsible for. The master type should be the same as his community.

Web services: the Web services populating the communities are created in the third step. Each Web service has a unique ID and is associated with only one community. Each Web service has its own QoS that is assigned at the beginning of the simulation.

After-RUN summary: Figure 3.3 shows the screenshot of the after-RUN summary. As each simulation consists of many runs, each run is concluded by a summary. In this figure, the number of feedback along with the new reputation value is shown (the reputation value is calculated for each community according to the equation 5, Subsection 3.2.2). The total positive refers to the positive feedback during the most recent run, and the total negative refers to the negative feedback during the most recent run.

Requests and ratings: Figure 3.4 illustrates the screenshot of service request and feedback rating. Here we can see the users while requesting for services from the communities with the feedback each user is giving to the community based on the service quality.

Ranking list: Figure 3.5 illustrates the screenshot of the ranking list. This figure shows the ranking of communities for each run. The ranking lists the communities with high reputation first and the list is updated after each run during the simulation.

This list is based on the reputation of each community since the starting of the simulation.

Reputation: Figure 3.6 depicts the screenshot of the most recent communities reputation. We can see in this figure the average of QoS each community provided in the run before.

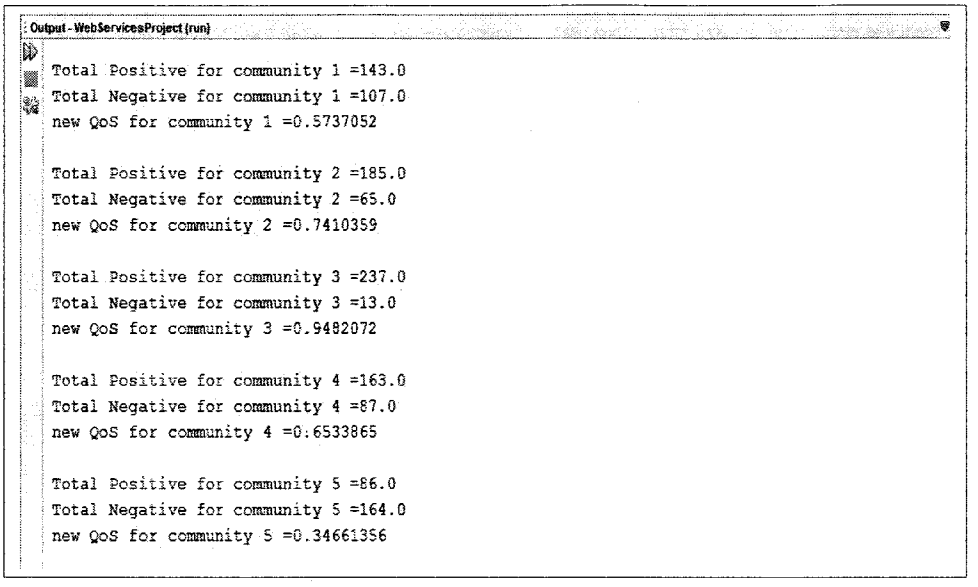


Figure 3.3: After-RUN summary

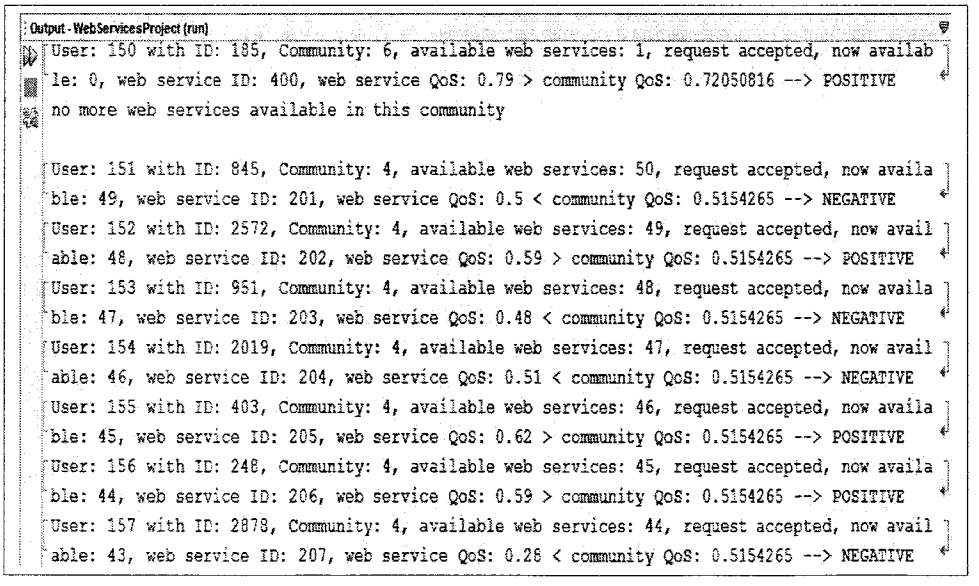


Figure 3.4: Requests and ratings

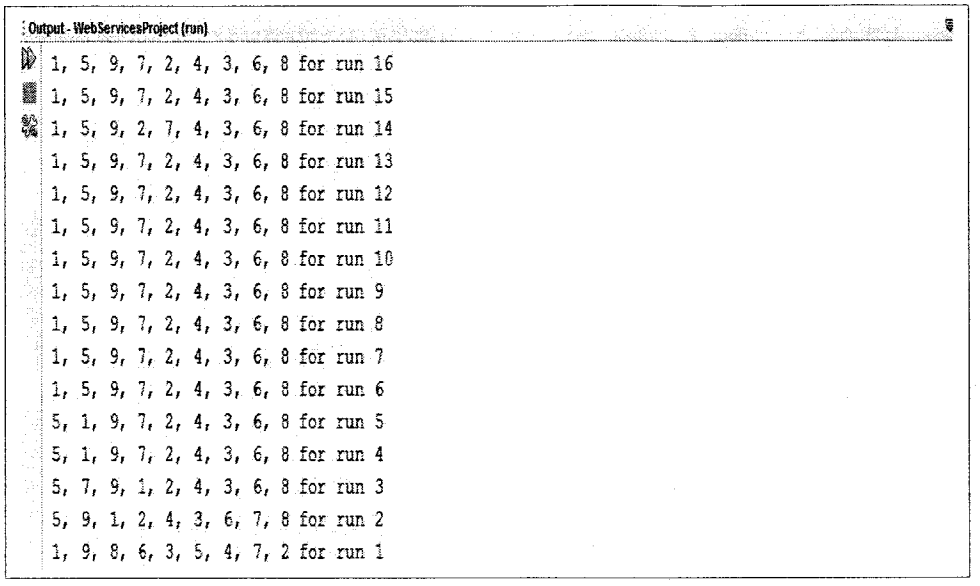


Figure 3.5: Ranking list

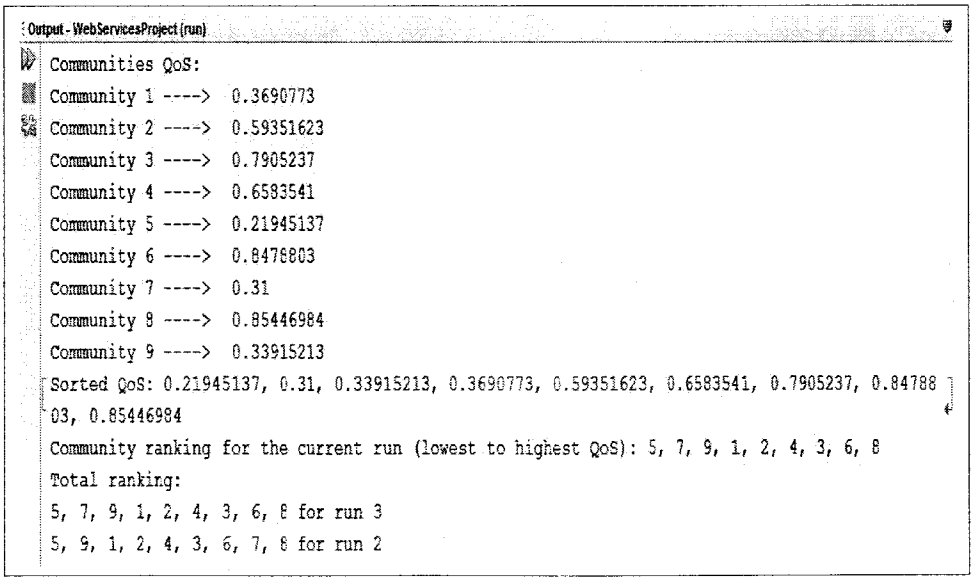


Figure 3.6: Reputation

3.5.2 Results and Analysis

This simulation consists of a series of empirical experiments tailored to show the adjustment of the communities reputation level. Table 3.1 represents three types of communities we consider in this simulation: ordinary, faker and intermittent. The classification is done according to the normal distribution function. Ordinary community (Figure 3.7-*Plot(a)*) acts normal and reveals what it has, the intermittent community (Figure 3.7-*Plot(b)*) is the one that alternatively changes its strategies over the time, and the faker community (Figure 3.7-*Plot(c)*) is the one that provides fake feedback in support of itself. As it is shown in table 3.1, the QoS value is divided into three ranges.

Table 3.1: Simulation summarization over the obtained measurements.

Community Type	Web service Type	Web service QoS
Ordinary	Good	[0.5, 1.0]
Faker	Bad	[0.0, 0.5]
Intermittent	Fickle	[0.2, 0.8]

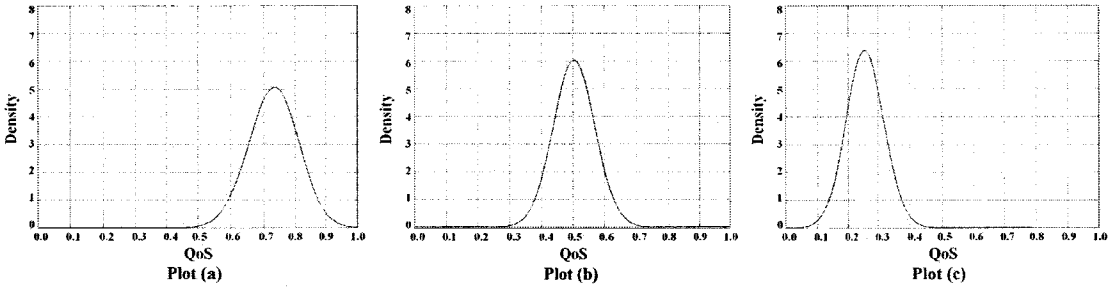


Figure 3.7: Communities classification

Over-RUN Reputation Level Analysis

In each RUN, a number of users are selected to search for the best service. Users are only directed to ask communities for a service and thus user would not find out about the Web service that is

assigned by the master of the community. In order to find the best community, the requesting user would evaluate the communities regarding to their reputation level.The selected community might be overloaded and consequently rejects the user requests. If the user is rejected from the best selected community, he would ask the second best community in terms of reputation level (and so on). After getting a response from a community, the user agent would provide a feedback relative to the quality of the obtained service and the community responsiveness. The feedback are logged in the logging mechanism that is supervised by the controller agent (Cg). The accumulated feedback would affect the reputation level of communities. In other words, the communities would loose their users if they receive negative feedback, by which their reputation level is dropped.

Considering the general incentive of communities to attract most possible users, communities in general, compete to increase their reputation level. Cheating on reputation level is done by colluding with some users to provide consecutive positive feedback in support of the malicious (faker) community.

In the empirical experiment, we are interested to observe the over-RUN reputation level of different types of communities and how efficient the adjustment is performed by the Cg. In the rest of this section, we will consider 2 communities: *Community 6,* which is an ordinary community; and *Community 8,* which is a faker community.

Figure 3.8 illustrates the plots of reputation level for a faker community $C8$. The *upper plot* represents the individual QoS for the community's assigned Web services. In this plot, the gray line defines the average QoS for the 50 Web services we have here. The most prominent feature of the plot is the comparison of the reputation level with the average of the community Web services QoS. The average value is assumed to be the actual QoS for the community and

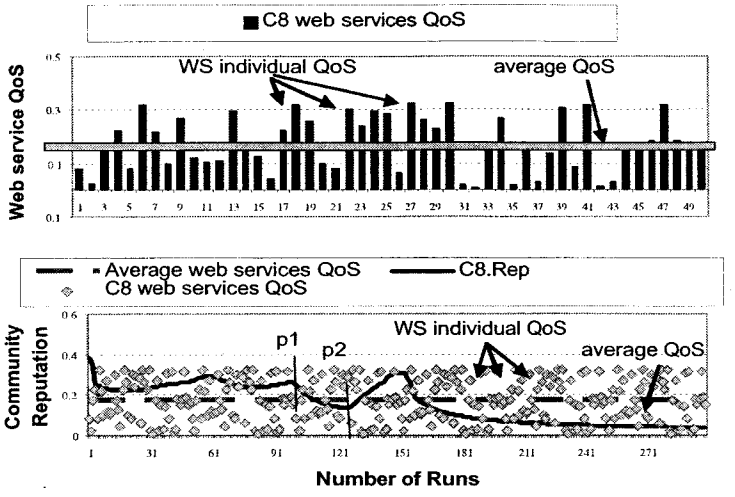


Figure 3.8: Communities overall quality of service vs. the number of simulation RUNs

thus, community's reputation level. In general, there would be convergence to such value if the community is acting in an ordinary manner (for $C8$ is 0.173). The *lower plot* illustrates the reputation level of this community over the 300 elapsing RUNs. Here we notify that the master of a community is responsible to assign the Web services to the user requests.

To this end, normally the high quality Web services are assigned first until they become unavailable, which forces the master agent to assign other lower quality Web services. Thus starting the RUNs, $C8$ gains reputation value (up to 0.313), which is better than its individual average quality of service. In figure 3.8 lower plot, the peek $P1$ defines the RUN in which the community $C8$ is out of high quality Web services. After passing this point, the reputation level of this community is decreased.

Figure 3.9 illustrates community $C8$ reputation level in comparison with an ordinary community $C6$. $C8$ at point $P3$ decides to provide fake positive feedback for himself to increase self reputation level. For the interval of 30 RUNs, this community gains higher reputation level up

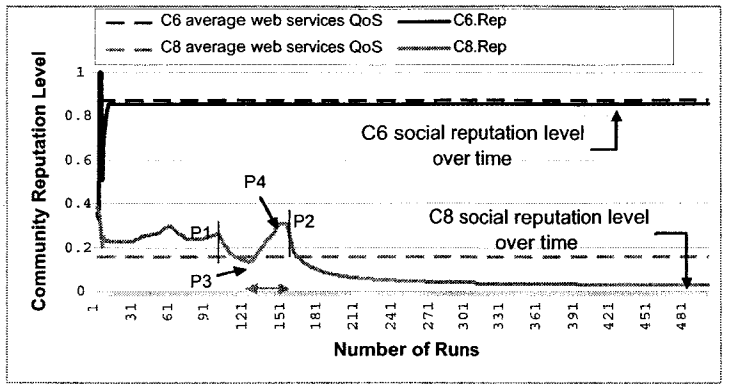


Figure 3.9: Communities overall quality of service vs. the number of simulation RUNs

to the point $P4$. The controller agent Cg, periodically verifies the feedback logs, in order to recognize the malicious actions. At $P4$ the controller agent Cg notices the malicious act of $C8$ and freezes the obtained feedback for investigation. Peek $P2$ is the point in which the community $C8$ is penalized in his reputation level. After $P2$, a drastic decrease in reputation value is seen which goes underneath $C8$'s average quality of service (up to 0.112). There is also a continuing but slower increase for the reputation of the faker community $C8$ that persists long after the first fake action recognition. Thus, there appear to be strong restriction effects, in which eventually the faker communities loose their users. However, there is also an ongoing effect of social influence, in which users doubt in communities that have drastic decrease in their reputation level.

Other simulation results on the reputation of communities of Web services versus single Web services are reported in Appendix A.

3.6 Summary

In this chapter, we proposed a new incentive-based reputation trust model for communities of Web services gathered to facilitate dynamic users requests. The reputation of the communities

are independently accumulated in binary feedback and stored in a log files reflecting the satisfaction of the users being serviced by the communities. The model represents a sound logging mechanism in order to maintain effective reputation assessment for the communities. The controller agent audits (investigates) the logging feedback files released by the users to preserve its integrity and to detect the fake feedback as a result of collusion between a community and a user (or a group of users), which are provided in support of the community. Upon detection, the controller agent maintains an adjustment in the logging system, so that the malicious community would be penalized in its reputation level. Our model has the advantage of providing a suitable metrics used to assess the reputation of a community. Moreover, having a sound logging mechanism, the communities would obtain the incentive not to act maliciously.

In this chapter also, and based on the observations on the experimental result of the simulations, we can conclude that the controller agent investigates the logging feedback released by the users to detect the fake feedback as a result of collusion between a community and a user (or a group of users), which are provided in support of the community. Upon detection, the controller agent maintains an adjustment in the logging system, so that the malicious community would be penalized in its reputation level.

Chapter 4

Game-Theoretic Analysis of Communities Reputation

Maintaining a sound reputation mechanism substantially requires a solid control and investigation. In this chapter, we propose a game-theoretic analysis of the reputation mechanism that we discussed in Chapter 3 to establish trust between users and services that are gathered into communities of Web services. In the architecture we proposed in Chapter 3, communities are ranked using their reputations as a result of provided feedback reflecting users satisfaction about the offered services. However, communities may alter their public reputation level by colluding with some users to provide fake feedback. In this chapter, game-theoretic analysis investigates the payoffs of different situations and elaborates on the facts that discourage malicious communities to fake feedback.

4.1 Game Theory

Game theory can be defined as the study of mathematical models of conflict and cooperation between intelligent rational agents. It gives the techniques for analyzing situations in which two or more agents make decisions that will influence one another's welfare [34]. To understand this definition we discuss what is meant by rational and intelligent agents.

Rational agent: is an agent that has clear preferences, models uncertainty via expected values, and always chooses to perform the action that results in the optimal outcome for itself from all feasible actions [43].

Intelligent agent: is an agent who knows everything about the game and can make any inferences about the situation [34].

4.1.1 Game Basics

A game in game theory consists of a set of agents (or players), a set of moves (or strategies) available to those players, and a specification of payoffs for each combination of strategies. It studies the interaction between agents whether it is cooperative or noncooperative. This concept can be formally defined as follows.

Definition 4.1. *A game in the normal form consists of the following basic elements:*

- *set of players* $P = \{1, 2, ..., n\}$,
- *set of strategy* S_i *for each player* i,

- *set of outcomes O which is also known as strategy profile and can be defined as:*

$$O = S_1 \times S_2 \times ...S_n$$

- *each player P_i has a utility function u_i, which is the preference over outcome and can be defined as:*

$$u_i : O \to \Re$$

Below we give an example of game and explain the aforementioned elements of the definition.

Example 4.1. *Consider a game with two players P_1 and P_2. Each player has the same set of strategies, thus $S_1 = S_2 = \{A, B, C\}$. In this case, each strategy is a single action. The utility functions are $u_1 : S_1 \times S_2 \longrightarrow \Re$ and $u_2 : S_1 \times S_2 \longrightarrow \Re$. For example, for any $X > 0$, these relations correspond to the following utilities: $u_1(A, C) = X$ and $u_2(A, C) = -X$ as shown to table 4.1*

Table 4.1: Payoff matrix.

		Player 2		
		A	B	C
Player 1	A	0,0	-X,X	X,-X
	B	X,-X	0,0	-X,X
	C	-X,X	X,-X	0,0

4.1.2 Solution Concepts

Dominant Strategy

For player i, the strategy s_i^* is the dominant strategy if it gives a greater payoff for any strategy profile of the remaining players in the game. Formally, dominant strategy is defined as:

$$u_i(s_i^*, s_{-i}) \geq u_i(s_i', s_{-i}), \forall_{s_{-i}}, \forall_{s_i'} \neq s_i^*$$

where s_{-i} is the strategy profile of all the players (agents) except player i.

Definition 4.2. *A strategy is strictly dominant for an agent if it strictly dominates any other strategy for that agent [25]*

Definition 4.3. *A strategy s_i is strictly dominated for an agent i if some other strategy s_i' strictly dominates s_i [25]*

Nash Equilibrium

A Nash equilibrium is an action profile s^* with the property that no player i can do better by choosing an action different from s_i^*, given that every other player adheres to s_{-i}^*. Formally, Nash equilibrium is defined as:

$$u_i(s_i^*, s_{-i}^*) \geq u_i(s_i', s_{-i}^*)$$

Theorem 4.1. *Every game with a finite number of players and action profiles has at least one Nash equilibrium [35]*

4.2 Game-Oriented Reputation Updates

The concept of reputation update is the fact of changing ones reputation level by which social opinions could be influenced. This concept simply forms a repeated game over time and consists of actions (made by communities) and reactions (made by Cg which is the agent that is assigned to monitor and audit the logging data) taking into consideration the architecture and the model we proposed in Chapter 3. For simplicity, we generalize acts of two parties in a period of $[t_0, t_2]$ where time t_0 is considered as the start time that community decides to whether fake and act maliciously (so-called F) or act as normal (so-called N). We suppose that $[t_0, t_2] = [t_0, t_1] \cup]t_1, t_2]$ where t_1 is the time that Cg recognizes suspicious feedback provided for the community. In this case, if the community has really acted maliciously, there would be an improvement in its reputation, which we compute here and refer to it henceforth as $Imp^{[t_1,t_2]}$:

$$Imp^{[t_1,t_2]} = Rep^{[t_b,t_2]} - Rep^{[t_b,t_1]}$$

Where $Rep_+^{[t_b,t_1]}$ is the overall reputation (can be computed as the average reputation) of the considered community during the period $[t_b, t_1]$ (t_b is the beginning time at which the community has been created). Time t_2 is the end of the typical period since Cg decides about the update in the reputation (denoted by $Rep_+^{[t_b,t_2]}$). At this time, Cg may decide to penalize (P^{t_2}) the community so that the adjusted reputation would be as follows:

$$\begin{aligned} Rep_+^{[t_b,t_2]} &= Rep^{[t_b,t_2]} - Imp^{[t_1,t_2]} - P^{t_2} \\ &= Rep^{[t_b,t_1]} - P^{t_2} \end{aligned}$$

If Cg ignores community's act, the adjusted reputation would be as follows:

$$Rep_{+}^{[t_b,t_2]} = Rep^{[t_b,t_2]}$$

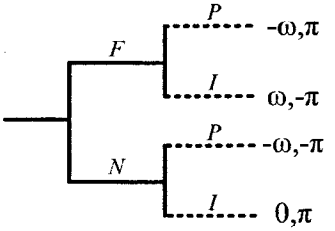
During the update process, Cg may make a mistake mentioned as false alarm that could be positive or negative. In the first case (false positive), the actual reputation (denoted by $Rep_{++}^{[t_b,t_2]}$) is more than the assigned reputation ($Rep_{+}^{[t_b,t_2]}$) and here the community looses as a result of Cg's mistake. In the second case (false negative), the actual reputation is less than the assigned value and here the community wins as a result of Cg's mistake. For both cases the difference (actual and assigned reputation level) is the same:

$$|Rep_{++}^{[t_b,t_2]} - Rep_{+}^{[t_b,t_2]}| = Imp^{[t_1,t_2]} + P^{t_2}$$

We refer to the aforementioned difference by ω ($\omega = Imp^{[t_1,t_2]} + P^{t_2}$) and consider it as the payoff of the game that could be either positive or negative. In the first case (false positive), considered community looses ω, which means its correct reputation level is dropped by this amount. In the second case (false negative), the considered community wins ω, which means its correct reputation level is increased by this amount.

4.3 Game Analysis

This section is dedicated to analyze the incentives and equilibria of reputation mechanism using the feedback logging system. Since the challenge is on the reputation (from community's point of view) and accuracy of the logging file (from Cg's point of view), we model the problem as a



Part (a): Payoff tree

	F	N
P	-ω,π	-ω,-π
I	ω,-π	0,π

Table (b)

Figure 4.1: One-shot game tree and table between community C and controller agent Cg.

two-player game between a typical community (C) and the controller agent (Cg).On one hand the strategy of faking (F) refers to the whole process of collusion and positive feedback that are submitted for self-support. The strategy of acting normal (N) introduces the state of no collusion and malicious act. On the other hand, Cg chooses between two strategies: penalizing (P) and ignoring (I) the event whether it is fake action or normal and not investigating the case. Here we define two utilities that are given to players C and Cg (respectively ω as computed in Section 4.2 and π). Obviously community C wins ($+\omega$) once Cg ignores the fake action, otherwise it looses ($-\omega$). Acting N causes no utility for C but impacts utility assigned for the controller agent. For Cg, positive utility is assigned once the recognition is right ($+\pi$), otherwise negative utility is assigned ($-\pi$). $+\pi$ ($-\pi$) represents then the increase (decrease) of the degree of Cg's accuracy in detecting malicious acts. The degree of accuracy (D_{ac}) is computed as follows:

$$D_{ac} = G_D/T_D$$

where G_D is the number of good detections and T_D is the total number of detections.

4.3.1 1-Shot Game Analysis

Figure 4.1-$Part(a)$ illustrates the tree of actions made by the two players. The leaves represent the outcome utility for the community and controller agent. Figure 4.1-$Table(b)$ gives the outcome table for this one-shot game. In pairs, the first value is assigned to C as player 1 and the second value is assigned to Cg as player 2. For example, if C plays F (fake) and Cg plays P (penalize), so the payoff of C is $-\omega$ and the payoff of Cg is $+\pi$. In general, C obtains ω once it fakes and being ignored. It gets $-\omega$ once being penalized (either by mistake or otherwise), and gets 0 once its normal act is not penalized.

Proposition 4.1. *In one-shot game, acting fake is the dominant strategy for the community.*

Proof. As shown by table (b) in Figure 4.1, by acting fake a better payoff is received by C whatever Cg's strategy is (I or P). □

Proposition 4.2. *In one-shot game, penalizing a fake action is the unique Nash equilibrium.*

Proof. Clearly acting fake by community C, controller agent Cg would have a best utility if penalizing strategy is chosen rather than ignoring. On the other hand, if Cg chooses to penalize, C would not change its chosen strategy since in both cases C will loose $-\omega$. Adversely, the normal act by C would let Cg to ignore. However, if the strategy is to ignore (by Cg), the best strategy for C is to act fake. Therefore, there is no Nash in ignoring the normal act. □

The unique Nash discussed in the previous proposition is a good situation for the controller agent, but this does not hold for community C. In this game, Cg gains payoff with respect to its accuracy in fake detection. However, C looses in any sort of penalizing. In fact, in one-shot game, strategies are adopted with respect to the available information that only reflect the current

state of both players. To this end, the best chosen strategies might end up with a dissatisfactory social situation. Let us now discuss what would be a socially better situation for the players in this game.

Definition 4.4. ***Pareto-Optimality.*** *A situation in a game is said to be Pareto-Optimal once there is no other situation, in which one player gains better payoff and other players do not loose their current payoff.*

Proposition 4.3. *In one-shot game, ignoring actions (F or N) by Cg are Pareto-Optimal.*

Proof. As it is clear from Figure 4.2-Table(b), by ignoring fake action, Cg receives $-\pi$ and this payoff could be increased only if C receives less than ω. Similar case is ignoring the normal act as there is no possible Pareto-improvements for both agents. □

The second Pareto-Optimal in this game (ignoring normal action) is a safer case in the sense that both players are at least non-negative. This is the best *social welfare* if being reached and is referred to as *Pareto-Optimal Socially Superior.*

From Proposition 4.1, we can conclude that faking would be the chosen strategy by C. In this case, C does not take into account the previous stages of the game and thus, does not learn to adopt the best strategy that maximizes its payoff. If C can estimate the expected payoff with respect to Cg's response, it might prefer acting normal. In fact, this issue is how to make agents (C and Cg) converge to the Pareto-Optimal Socially Superior as a rational choice [3]. In the following, we extend the one-shot game to the repeated game over periods that we typically refer to as $[t_0, t_2]$. We analyze the same game with respect to strategies and corresponding payoffs that are assigned such that the two players have more various range of payoffs. At time t_1, Cg will decide whether to continue or stop investigating. To this end, e_0 is referred to as the case of

no effort is made in doing investigation and basically ignoring all actions. Otherwise, the best effort is made by Cg doing investigation. Therefore at time t_2, Cg decides about its strategy. Obviously, if Cg chooses e_0 and C plays fake, controller agent will loose right away. We split the game time to two time intervals of $[t_0, t_1]$ and $[t_1, t_2]$ and strategy of acting in each interval needs to be decided. We apply a weight to each time interval regarding its importance (also the payoff portion). Consider μ as the payoff coefficient for the acts done in $[t_0, t_1]$ and $1 - \mu$ as the payoff coefficient for the acts done in $[t_1, t_2]$.

4.3.2 N-Shots Game Analysis

For simplicity and illustration purposes but without losing generality, we consider the repeated game with two-shots. The general case with n-shots ($n \geq 2$) will follow. In such a game, community C as player 1 has two information sets. The first set contains decisions over fake F and act normal N given that community is in the first decision time spot (μ). The second set contains decisions over fake and act given that community is in the second decision time spot ($1-\mu$). Since community C is the game starter, and controller agent Cg initially decides whether to stop or continue the game, we consider two continuous actions that reflect our game the best. Therefore, community's set of strategies is as follows:

$$S_C = \{F^{\mu}F^{1-\mu}, F^{\mu}N^{1-\mu}, N^{\mu}F^{1-\mu}, N^{\mu}N^{1-\mu}\}$$

In the general case with n-shots, we will have:

$$S_C = \{F^{\mu_1} \dots F^{\mu_n}, F^{\mu_1} \dots N^{\mu_n}, \dots, N^{\mu_1} \dots N^{\mu_n}\}$$

Table 4.2: Two-shots game of community C and controller agent Cg with obtained payoffs.

Controller agent Cg \ Community C	$F^{\mu}F^{1-\mu}$	$F^{\mu}N^{1-\mu}$	$N^{\mu}F^{1-\mu}$	$N^{\mu}N^{1-\mu}$
e_0	ω,-π	μω,-μπ	(1-μ)ω,-(1-μ)π	0,0
$P^{\mu}P^{1-\mu}$	-ω,π	-ω,(2μ-1)π	-ω,(1-2μ)π	-ω,-π
$P^{\mu}I^{1-\mu}$	(1-2μ)ω,(2μ-1)π	-μω,π	(1-2μ)ω,-π	-μω,(1-2μ)π
$I^{\mu}P^{1-\mu}$	(2μ-1)ω,(1-2μ)π	(2μ-1)ω,-π	-(1-μ)ω,π	-(1-μ)ω,(2μ-1)π
$I^{\mu}I^{1-\mu}$	ω,-π	μω,(1-2μ)π	(1-μ)ω,(2μ-1)π	0,π

where $\sum_{i=1}^{n} \mu_i = 1$. Considering the choice of efforts, Cg's set of pure strategies (penalizing P or ignoring I) is as follows:

$$S_{Cg} = \{e_0, P^{\mu}P^{1-\mu}, P^{\mu}I^{1-\mu}, I^{\mu}P^{1-\mu}, I^{\mu}I^{1-\mu}\}$$

Table 4.2 represents the payoff table of the two players over their chosen strategies. We continue our discussions in the rest of this section on this table.

Proposition 4.4. *In repeated game, faking all the time and penalizing all fake actions is the unique Nash equilibrium.*

Proof. (We illustrate the proof for two-shots game from which the general case follows.) **1) Nash:** It is clear from Table 4.2 that in both faking intervals, Cg receives the maximum payoff by penalizing both cases. In this case, C will not increase its payoff ($-\omega$) and thus, will not prefer any other strategy. In any other case, by choosing the maximum received payoff for any player, the other player has a better strategy to increase its payoff. **2) Uniqueness:** We prove that the Nash equilibrium is the only Nash with respect to the following discussions. In the first row of Table 4.2, there is no Nash because Cg makes no effort, so maximum received payoff is

zero and thus, any time the preference is to change to a state that effort is made to penalize fake actions. In third and forth rows, still there is no Nash since in these rows there are choices of P and I in the sense that for any of these choices, C will gain more by selecting a that maximizes its assigned payoff. In the last row, the payoff assignment is similar to the first one, so that Cg prefers to change its chosen strategy to apply penalty to fake actions. □

We also have the following propositions generalized from the two-shots game:

Proposition 4.5. *In repeated game, faking all the time is dominant strategy for C.*

Proposition 4.6. *In repeated game, acting normal by C and ignoring by Cg all the time is Pareto-Optimal Socially Superior.*

The analysis of the obtained Nash equilibrium is important in the sense that the stable point is not a socially preferred situation for both players. Indeed, Cg has a challenge of not doing wrong. To this end, any time a fake action is recognized, best payoff is received once Cg applies penalties. On the other hand, for rational community C, the best strategy is to fake given that ignore by Cg is ensured. Similar to one-shot game, choosing fake is dominant strategy for C. This fact holds only if C considers Cg's strategies with equal probabilities. However, these probabilities are different and if C is aware of them, the strategy chosen by C would be different. In general, the probability of correct recognition by Cg totally impacts the strategy that C adopts in the repeated game. Thus, we need to consider the recognition probabilities of Cg $(q^{t_0}, \ldots, q^{t_n})$ that simply reflect its accuracy evolving over time. The increasing of Cg's accuracy is reflected by the fact that $q^{t_0} \leq \cdots \leq q^{t_n}$. Indeed, Cg's accuracy has impact on expected reputation that community C estimates given the penalty and improvement it makes. Therefore, Cg applies such penalty that discourages C to act fake.

Proposition 4.7. *If* $P^{t_n} > \frac{1-q^{t_n}}{q^{t_n}} Imp^{[t_{n-1},t_n]}$, *then community* C *receives less reputation value if it acts fake* [1].

Proof. To prove the proposition, we consider the condition true and prove that $E(Rep_{++}^{[t_b,t_n]}|(N/F)^{\mu_1} \dots (N/F)^{\mu_n}) \leq E(Rep_{++}^{[t_b,t_n]}|N^{\mu_1} \dots N^{\mu_n})$. By simple calculation we expand the expected value to its possible cases together with their probabilities, so we get:

$$E(Rep_{++}^{[t_b,t_n]}|(N/F)^{\mu_1} \dots (N/F)^{\mu_n}) =$$
$$(q^{t_n})(Rep^{[t_b,t_n]} - Imp^{[t_{n-1},t_n]} - P^{t_n}) + (1 - q^{t_n})(Rep^{[t_b,t_n]})$$

this is the expected reputation level given that a fake action is made in any of the intervals.

$$E(Rep_{++}^{[t_b,t_n]}|N^{\mu_1} \dots N^{\mu_n}) = Rep^{[t_b,t_n]} - Imp^{[t_{n-1},t_n]}$$

and this is the expected reputation level given that no fake action is made during any of the interval.

$$E(Rep_{++}^{[t_b,t_n]}|(N/F)^{\mu_1} \dots (N/F)^{\mu_n}) - E(Rep_{++}^{[t_b,t_n]}|N^{\mu_1} \dots N^{\mu_n}) \geq 0$$

Considering $P^{t_n} \geq \frac{1-q^{t_n}}{q^{t_n}} Imp^{[t_{n-1},t_n]}$, the difference of the obtained values is positive, so we are done. □

In proposition 4.7, the required penalty is measured that accomplish reputation decremented manner. We rewrite this inequality in terms of Cg accuracy ($q^{t_n} \geq \frac{1}{Imp^{[t_{n-1},t_n]}+P^{t_n}} = \frac{1}{\omega}$). The obtained relation highlights the dependency of the received payoff (ω) to the recognition probability of Cg. Therefore, in repeated game, if q increases (as a result of more experience and learning over time), ω would decrease, which reflects two facts: 1) C's less tendency to fake, which in

[1] *This proof is inspired by the proof done in [23]*

general applies to all active communities; and 2) Cg gets higher probability of penalizing since faking history is taken into account. Therefore, over time C tends to act normal.

Theorem 4.2. *In the repeated game with n-shots, if* $P^{t_n} \geq \frac{1-q^{t_n}}{q^{t_n}} Imp^{[t_{n-1},t_n]}$, *acting normal and being ignored is both Nash and Pareto-Optimal.*

Proof. Considering the proposition 4.6, ignoring normal intervals (or zero effort in normal intervals) are Pareto-Optimal. On the other hand, deduced from proposition 4.7, C would have less reputation if it fakes given that it is aware of the assigned penalty and Cg's accuracy. Therefore, the dominant strategy for C would be acting N. If C plays N as its dominant strategy, the best response from Cg would be I in all intervals. This state is a healthy state that both players would be stable. This state would be Nash once the condition expressed in Proposition 4.7 holds. In this case, $N^{\mu_1} \ldots N^{\mu_n}$ and $I^{\mu_1} \ldots I^{\mu_n}$ are dominant strategies for C and Cg. □

Considering the high accuracy of Cg that is maintained over time, C has a very low chance to fake and receive a positive payoff. Therefore, a safer strategy is to act normal by which a non-negative payoff is guaranteed. No payoff reflects no change in reputation level, which makes communities acting normal and receiving positive feedback that are provided for them by the users.

4.4 Experimental results

In this section we describe the simulations we perform on the game-theoretic analysis of the reputation mechanism. Here we adapt the same simulation environment we have introduced in Chapter 3.

4.4.1 Experiment

In this chapter, we have proved that fake action will eventually end up with a penalty that is totally not preferable by any community and acting normal is Pareto-Optimal in repeated game. In this section, we demonstrate the aforementioned facts using simulation of the environment populated with different types of agent-based communities and users.

In the simulation, a sequence of RUNs have been examined to highligh the aforementioned system parameters in more details. In Table 4.3, we provide three types of communities we deploy in simulation: ordinary, faker and intermittent. Ordinary community acts as normal and offers its truth quality of service (QoS) that could be in any range. In maximum overloaded time, this type of community responses (Rs) 20% on top of its capacity and manages to provide the service with 80% accuracy (Ac). Faker community populates 40% of communities and normally have a relatively lower QoS. This type of community takes 50% over its capacity and offers service with 40% of accuracy. Intermittent communities might change their strategies with respect to the environment and their accuracy is 60%. In Table 4.3, we also represent different Web service types that are active once they belong to a community. A good Web service offers more than average QoS and handles 10 requests at a time. However, bad and fickle web services handle more requests with a relatively lower QoS.

In each RUN, a number of users are selected to search for the best service. Strictly speaking, users are only directed to ask community for a service and thus, user would not find out about the Web service that is assigned by the master of the community. In order to find the best community, the requesting user would evaluate the communities regarding their reputation level. Some times,

Table 4.3: Simulation summarization over the obtained measurements.

Community Type	Density	QoS	Rs	Ac
Ordinary	30.0%	[0.0%, 100.0%]	20.0%	80.0%
Faker	40.0%	[0.0%, 60.0%]	50.0%	40.0%
Intermittent	30.0%	[20.0%, 70.0%]	30.0%	60.0%

Web service Type	Density	QoS	Capacity
Good	30.0%	[50.0%, 100.0%]	10
Bad	30.0%	[0.0%, 50.0%]	20
Fickle	40.0%	[20.0%, 80.0%]	20

users are in contact with some communities that used to offer acceptable QoS, so the users re-select them. The selected community might be highly overloaded and consequently rejects the user requests. If the user is rejected from the best selected community, he would ask the second best community in terms of reputation level (and so on). The feedback are logged in the logging mechanism that is supervised by controller agent Cg. The accumulated feedback would affect the reputation level of communities. In other words, the communities would loose their users if they receive negative feedback, by which their reputation level is dropped.

One of the main issues discussed in this game-theoretic study is the total reputation of communities that are rated through provided feedback and adjusted via Cg's inspections over time. In general, a reputation of a typical ordinary community (C_o) tends to approach its actual QoS. This simply means that its truth service offering (with 80% accuracy) would cause C_o to receive feedback that bring its total reputation as high as its actual QoS. Figure 4.2 illustrates such approach in the left plot. The bold curve represents C_o's reputation that approaches its QoS over time. The dotted curve reflects a similar community C'_o that has been penalized by Cg (obviously by mistake). In this plot, different reputation levels have been shown, which reflects possibilities of obtained reputation for an ordinary community over time. The right plot is similar and illustrates reputation adjustment of a faker community (C_f) that has been penalized and thus, decreased

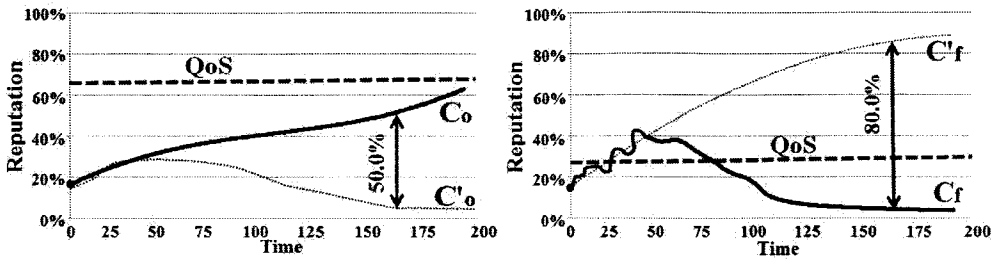


Figure 4.2: Communities's total reputation adjustment possibilities vs. simulation RUNs.

reputation level is set. Likewise, the dotted curve reflects the case that C'_f is not recognized and thus, a high faked reputation is obtained. In the left plot, Cg's mistake causes 50% of reputation adjustment, whereas in the right plot that is 80%. This highlights the fact that ignoring a faker community (C'_f) would distract the system more than penalizing an ordinary community (C'_o). Nevertheless, for any type of false detection, the accuracy of Cg decreases and therefore, Cg aims to maintain the highest possible accuracy level.

Controller agent Cg's accuracy is also a challenging issue in our discussions. The importance inspired from the fact that once Cg puts the minimum effort in inspection, the best strategy maximizing community's payoff is to totally fake. Even if the effort is made by Cg, the lower accuracy would cause ignoring malicious acts and thus, a potential payoff is generated for a faker community. To this end, Cg's accuracy level and its effort on inspecting feedback affects communities' tendencies to fake. Figure 4.3 in the left illustrates Cg's overall accuracy over time together with percentage of communities (faker and intermittent) that tend to take advantage of Cg's inconsistency. As it is obvious from the plot, the percentage of communities that consider faking feedback is dramatically decreasing once the accuracy is getting higher. In general, Cg aims to increase the accuracy and that is manageable once a certain number of RUNs is passed and Cg gets to know the environment. In addition, recognizing a faker community at any time would

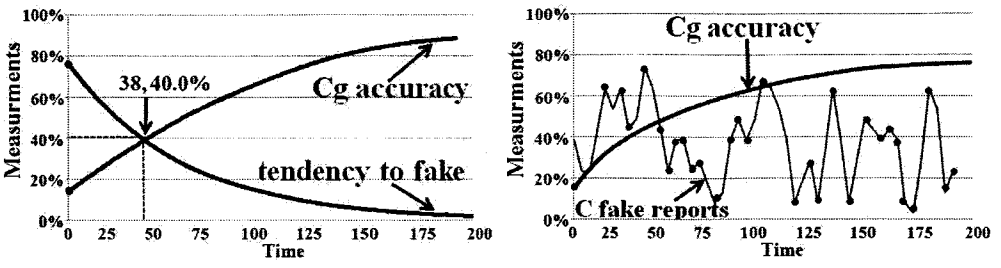


Figure 4.3: Controller agent's accuracy measurements vs. simulation RUNs.

cause a higher caution in future. Therefore, inspections are more detailed that cause a higher accuracy. The tendency is dramatically decreasing because penalties are getting higher for the second fake recognition (and so on). To this end, the percentage of communities that tend to fake is decreasing but Cg is still subject to falsely penalize (not malicious acts) and that prevents 100% accuracy in any time. In general, Cg's accuracy in detection is partially related to strategies that faker or intermittent communities take. For instance, Cg does not expect an immediate second fake from a penalized community because they are assumed to act rationally. But if communities act maliciously by random and do not care about the penalty they may obtain, Cg's accuracy would get lower. This is shown in Figure 4.3 in the right plot. In this plot, the random faking strategies would cause the percentage of fake actions to oscillate with nonpredictable manner.

4.5 Conclusion

In this chapter, we discussed the two-players game analysis over the reputation-based architecture that hosts communities of Web services. In the deployed architecture, communities can collude with users to increase self reputation. Meanwhile, controller agent investigates user feedback and penalizes malicious communities. Controller agent may fail to accurately function, which is

known as incentive for some communities to act maliciously. The discussion is formed in terms of a game that is analyzed in one-shot and repeated cases. This analysis is concluded by denoting the best social state in which communities are discouraged to act maliciously and increase self reputation.

In this chapter also, and based on the observations of the experimental results of the simulations, we can conclude that the controller agent may fail to accurately function. This could encourage some communities to act maliciously. In the conducted experiments, malicious communities are observed and their characteristics are measured over time. In general, best Pareto-Optimal is observed to be a stable state for communities and controller agent.

Chapter 5

Conclusion and Future Work

The primary goal of the thesis was to develop a framework to solve the problem of reputation of Web services communities as outlined in Chapter 1. That goal has been achieved with a new reputation model based on the logging mechanism we introduced in Chapter 3. The method has been implemented and evaluated using a variety of experiments. Another goal was to maintain the reputation mechanism from malicious behavior. This is accomplished by the use of game-theoretic analysis as detailed in Chapter 4 to establish trust between the communities of Web services and the users. The game-theoretic analysis allows the controller agent Cg to follow strategies that help in controlling and investigating the logging mechanism, which in turn help in maintaining a sound reputation mechanism.

5.1 Summary of Contributions

In this thesis, we presented two main contributions to the reputation of communities of Web services. The first contribution of this thesis is the proposition of a new incentive-based reputation model for communities of Web services gathered to facilitate dynamic users requests. The reputation of the communities are independently accumulated in binary feedback reflecting the satisfaction of the users being served by the communities. A model representing a sound logging mechanism in order to maintain effective reputation assessment for the communities has been discussed. The controller agent Cg investigates the logging feedback released by the users to detect the fake feedback as a result of collusion between a community and a user (or a group of users), which are provided in support of that community. Upon detection, the controller agent maintains an adjustment in the logging system, so that the malicious community would be penalized by reducing its reputation level. Our model has the advantage of providing suitable metrics used to assess the reputation of a community. Moreover, having a sound logging mechanism, the communities would obtain the incentive not to act maliciously. The proposed mechanism efficiency is analyzed through a defined testbed.

The second contribution of this thesis is the analysis over the reputation-based infrastructure that hosts communities of Web services as providers of services, users as consumers of services, and controller agent Cg as reputation manager in the system. In the deployed infrastructure, communities can collude with users to increase self reputation. Meanwhile, controller agent investigates user feedback and penalizes malicious communities. Controller agent may fail to accurately function, which is known as incentive for some communities to act maliciously. The discussion is formed in terms of a game that is analyzed in one-shot and repeated cases. This

analysis is concluded by finding the best social state in which communities are discouraged to act maliciously and increase self reputation. The analysis is accompanied by empirical results that highlight reputation system's parameters. In experimental results, malicious communities are observed and their characteristics are measured over time. In general, the best Pareto-Optimal is observed to be a stable state for communities and the controller agent.

5.2 Future Work

Although this thesis has answered how reputation is relevant to build trust in online environment of communities of Web services, it opens up more research opportunities and questions that are unanswered. This section describes a few of these important issues:

- Advance the assessment model to enhance the model efficiency using the comprehensive approach that was developed in [24], which considers the trust issue as an optimization problem. In the logging system, we need to optimize detection process, trying to formulate it in order to be adaptable to diverse situations.

- Take advantage of the autonomous agents self-learning capabilities to make better decisions in future interactions. Conducting more transactions with whom they have a good history, and have a black list of dishonest Web services.

- For our model, the initial reputation values distribution is based on normal distribution function, this can be extended where reputation bootstrapping can be applied [28].

- Advance the game theoretic discussion such that communities that risk the malicious act deploy a learning algorithm that enables them to measure their winning chance. To this

end, a continuous game can be extended that both players update their selected policies. Similarly, we need to discuss more about controller agent's different false detection cases that distract reputation management.

Bibliography

[1] Ali Shaikh Ali, Simone A. Ludwig, and Omer F. Rana. A cognitive trust-based approach for web service discovery and selection. In *ECOWS '05: Proceedings of the Third European Conference on Web Services*, 2005.

[2] Gustavo Alonso. *Web services: concepts, architectures and applications*. Springer, 2004.

[3] Dipyaman Banerjee and Sandip Sen. Reaching pareto-optimality in prisoner's dilemma using conditional joint action learning. *Autonomous Agents and Multi-Agent Systems*, 15 (1):91–108, 2007.

[4] Jamad Bentahar, Zakaria Maamar, Wei Wan, Djamal Benslimane, Philippe Thiran, and Sattanathan Subramanian. Agent-based communities of web services: an argumentation-driven approach. *Service Oriented Computing and Applications*, 2(4):219–238, 2008.

[5] Jamal Bentahar, Zakaria Maamar, Djamal Benslimane, and Philippe Thiran. An argumentation framework for communities of web services. *IEEE Intelligent Systems*, 22(6):75–83, 2007.

[6] David A. Chappell and Tyler Jewell. *Java Web services*. O'Reilly Media, Inc., 2002.

[7] NetBeans Community. Netbeans ide 6.5 beta download. http://download.netbeans.org/netbeans/6.5/beta/, 2008. viewed August 02 2008.

[8] Chrysanthos Dellarocas. The digitization of word of mouth: Promise and challenges of online feedback mechanisms. *Management Science*, 49(10):1407–1424, 2003.

[9] Said Elnaffar, Zakaria Maamar, Hamdi Yahyaoui, Jamal Bentahar, and Philippe Thiran. Reputation of communities of web services - preliminary investigation. In *AINAW '08: Proceedings of the 22nd International Conference on Advanced Information Networking and Applications - Workshops*, pages 1603–1608, 2008.

[10] Peyman Faratin, Carles Sierra, and Nick R. Jennings. Negotiation decision functions for autonomous agents. *International Journal of Robotics and Autonomous Systems*, 24(3 - 4): 159–182, 1998.

[11] Organization for the advancement of structured information standards. Introduction to uddi: Important features and functional concepts, 2004.

[12] Elodie Fourquet, Kate Larson, and William Cowan. A reputation mechanism for layered communities. *SIGecom Exch.*, 6(1):11–22, 2006.

[13] W3C Working Group. Web services architecture. http://www.w3.org/TR/ws-arch/, February 2004. viewed October 11 2009.

[14] James Hendler. Agents and the semantic web. *IEEE Intelligent Systems*, 16(2):30–37, 2001.

[15] Mariusz Jacyno, Seth Bullock, Michael Luck, and Terry R. Payne. Emergent service provisioning and demand estimation through self-organizin agent communities. In *The Eighth International Conference on Autonomous Agents and Multiagent Systems*, 2009.

[16] Nicholas R. Jennings. Agent-based computing: Promise and perils. In *In Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 1429–1436, 1999.

[17] Radu Jurca and Boi Faltings. An incentive compatible reputation mechanism. *E-Commerce Technology, IEEE International Conference on*, 0:285–292, 2003.

[18] Radu Jurca and Boi Faltings. Reputation-based service level agreements for web services. In *Service-oriented computing: ICSOC 2005: third international conference, Amsterdam, The Netherlands, December 12-15*, 2005.

[19] Radu Jurca and Boi Faltings. Obtaining reliable feedback for sanctioning reputation mechanisms. *Journal of Artificial Intelligence Research*, 29(1):391–419, 2007.

[20] Radu Jurca, Boi Faltings, and Walter Binder. Reliable qos monitoring based on client feedback. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, 2007.

[21] Georgia Kastidou, Robin Cohen, and Kate Larson. A graph-based approach for promoting honesty in community-based multiagent systems. In *8th International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems*, 2009.

[22] Georgia Kastidou, Kate Larson, and Robin Cohen. Exchanging reputation information between communities: A payment-function approach. In Craig Boutilier, editor, *IJCAI*, pages 195–200, 2009.

[23] Babak Khosravifar, Jamal Bentahar, Philippe Thiran, Ahmad Moazin, and Adrien Guiot. An approach to incentive-based reputation for communities of web services. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*, pages 303–310, 2009.

[24] Babak Khosravifar, Maziar Gomrokchi, Jamal Bentahar, and Philippe Thiran. Maintenance-based trust for multi-agent systems. In *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 1017–1024, 2009.

[25] Kevin Leyton-Brown and Yoav Shoham. *Essentials of Game Theory: A Concise, Multidisciplinary Introduction (Synthesis Lectures on Artificial Intelligence and Machine Learning)*. Morgan and Claypool Publishers, 2008. ISBN 1598295934.

[26] Yutu Liu, Anne H. Ngu, and Liang Z. Zeng. Qos computation and policing in dynamic web service selection. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 66–73, 2004.

[27] Zaki Malik and Athman Bouguettaya. Evaluating rater credibility for reputation assessment of web services. In *Web Information Systems Engineering*, pages 38–49, 2007.

[28] Zaki Malik and Athman Bouguettaya. Reputation bootstrapping for trust establishment among web services. *IEEE Internet Computing*, 13(1):40–47, 2009.

[29] Anne Thomas Manes. *Web Services: A Manager's Guide*. Addison-Wesley Professional, 2003.

[30] Umardand Shripad Manikrao and T. V. Prabhakar. Dynamic selection of web services with recommendation system. In *NWESP '05: Proceedings of the International Conference on Next Generation Web Services Practices*, 2005.

[31] E. Michael Maximilien and Munindar P. Singh. Conceptual model of web service reputation. *SIGMOD Rec.*, 31(4):36–41, 2002.

[32] E. Michael Maximilien and Munindar P. Singh. Multiagent system for dynamic web services selection. In *AAMAS Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE)*, 2005.

[33] Kirk McKusick. Interview with adam bosworth. *Queue*, 1(1):12–21, 2003.

[34] Roger Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, 1997.

[35] John Nash. Non-cooperative games. *The Annals of Mathematics*, 54(2):286–295, 1951.

[36] Eric Newcomer. *Understanding Web Services: XML, Wsdl, Soap, and UDDI*. Addison-Wesley Professional, 2002.

[37] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. Reputation systems. *Commun. ACM*, 43(12):45–48, 2000.

[38] Wanita Sherchan, Shonali Krishnaswamy, and Seng Wai Loke. Relevant past performance for selecting web services. In *QSIC '05: Proceedings of the Fifth International Conference on Quality Software*, pages 493–445, 2005.

[39] Sattanathan Subramanian, Philippe Thiran, Zakaria Maamar, and Djamal Benslimane. Engineering communities of web services. In *iiWAS*, volume 229, pages 57–66, 2007.

[40] W3C. Web services description language (wsdl) 1.1. http://www.w3.org/TR/wsdl.html, March 2001. viewed October 02 2009.

[41] W3C. Soap version 1.2 part 1: Messaging framework (second edition). http://www.w3.org/TR/soap12-part1/, April 2007. viewed October 02 2009.

[42] W3C. Extensible markup language (xml) 1.0 (fifth edition). http://www.w3.org/TR/2008/REC-xml-20081126/, November 2008. viewed October 02 2009.

[43] Wikipedia. Rational agent. http://en.wikipedia.org/wiki/Rational_agent, October 2009. viewed October 29 2009.

[44] Zhengping Wu and Alfred C. Weaver. Using web service enhancements to bridge business trust relationships. In *PST '06: Proceedings of the 2006 International Conference on Privacy, Security and Trust*, pages 1–8, 2006.

[45] Ziqiang Xu, Patrick Martin, Wendy Powley, and Farhana Zulkernine. Reputation-enhanced qos-based web services discovery. In *IEEE International Conference on Web Services, 2007. ICWS 2007*, pages 249–256, 2007.

Appendix A

Empirical Observations and Analysis of Communities versus Single Web Services

Having a high reputation value will eventually bring high demand from users and results in overloading service providers beyond their capacities for handling these requests. The challenge is to identify a tradeoff between one's capacity (maximum number of service offering at a time) and market share (number of users that request services) so that an efficient service can handle the requests in a way that neither it gets overloaded quickly, nor it remains idle. Therefore, the objective of all Web services is to tackle such a tradeoff in which the service gains a stable reputation and market share level. The motivation here is to observe experimentally in an environment populated with both single Web services and communities of Web services the better situation and hence the reputation a Web service can have. In this context, we consider the $InDemand$ value (InD_C and InD_S) for community and single Web service respectively, to be the number of all requests made by users to all communities and single Web services in the simulation. We also consider in this simulation the *capacity* value which is the maximum number of requests a

Table A.1: Environment summarization over the obtained measurements.

Type	Density	QoS	Capacity
Communities of Web services	40.0%	[20.0%, 80.0%]	[100,300]
Single Web Services	60.0%	[20.0%, 95.0%]	[10,20]

community (Cap_C) or a single Web service (Cap_S) can handle at the same time, for example a community with $Cap_C = 100$ can handle 100 requests simultaneously.

In this appendix, we provide an empirical analysis over the observed results regarding characteristics of a typical community (C) and a single Web service (S). We motivate the join option by depicting the challenges that a single Web service S faces when it cannot handle further requests because of capacity limitation ($InD_S^{[t_0,t_1]} > Cap_S$). The simulation consists of a series of empirical experiments tailored to show different parameters of system components in diverse aspects. Table A.1 summarizes the simulated environment which is populated with users and providers. Users are multiple and scattered over the environment, but providers are divided into communities and Single Web Services. Communities cover 40.0% of providers while Single Web Services are more (60.0%). In the simulated environment, we deployed relatively lower quality of service for communities to motivate their higher performance from their request handling rather than solely their service qualities. In general, communities host at least 10 Web services, which covers at least 100 requests at a time, while a single Web service handles 10 at a time.

One of the main reasons that distracts service provider's overall performance (P_S for single Web service or P_C for community) is its reputation update range. Since they are associated with a reputation level as a result of provided feedback by the users, if the number of interacting users is relatively low, the update over the reputation rank would be more visible than the case when the number of interacting users is relatively high. Figure A.1 shows characteristics of community (C) in the left part and single Web service S in the right part. Plots reflect average values that have

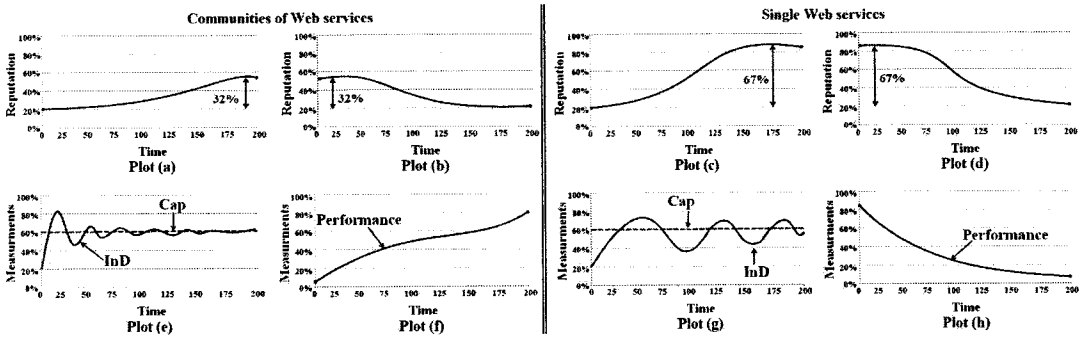


Figure A.1: Characteristics of communities vs. single Web services.

been measured as a result of analysis over all communities of the same type. $Plots(a)$ and (c) respectively represent the reputation increase for the same number of RUNs while both C and S are gaining high reputation. Since number of interacting users with S is lower than that of C, the effect that positive feedback make on the reputation value of S is relatively high compared to the effect on C. Similarly reputation degradation ($plot(b)$ for C and $plot(d)$ for S) shows dramatic change in single Web service compared to community. Overall, such high range of change reflects lower feedback density, which also reflects lower market share ($InD_S^{[t_0,t_1]} < InD_C^{[t_0,t_1]}$). This is normal since single Web service would share a small portion of the market. We elaborate on the effect of such a range more in $plots(e)$ and (g). In these plots, the horizontal line represents a community's capacity. What is interesting in these plots is the way that inDemand approaches capacity. In $plot(e)$, we observe closer oscillating curve compared to that in $plot(g)$. This is due to the fact that C handles its user increase until it gets overloaded ($InD_C^{[t_0,t_1]} > Cap_C$) and starts to offer lower quality services. Therefore, being overloaded, C gets some negative feedback that cause a drop in its inDemand. Once handling the requests again ($InD_C^{[t_0,t_1]} < Cap_C$), C provides quality services and obtains good feedback that increase its inDemand again ($InD_C^{[t_1,t_2]}$). But not necessarily C would obtain the same inDemand as before ($InD_C^{[t_1,t_2]} < InD_C^{[t_0,t_1]}$), and that

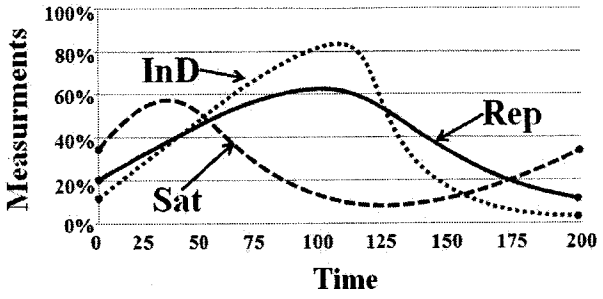


Figure A.2: Evolution of cooperative parameters for a single Web service over time.

is simply as a result of users' evaluations that might not show interest requesting C again that offered a low quality before. In general, inDemand value would be relaxed by capacity and that is the point where C handles user requests at best thanks to collaboration between Web services that share users. Such relaxation would take much longer time with S and that is due to its lower number of interacting users. As shown in $plot(g)$, the inDemand curve oscillates in a higher range and takes more time to merge with S's capacity level. We also show this fact with performance parameter in $plots(f)$ and (h). As shown in $plot(f)$, C as a good community obtains an interesting performance over time while S (see $plot(h)$) gets decreased. Performance parameter can be considered as obtained utility (or payoff) as a result of acting either alone or with other Web services in a community.

We continue our discussions in more details by comparing how the aforementioned parameters (inDemand and Capacity) evolve over time. In Figure A.2, characteristic of a typical Web service is measured to observe its cooperative parameters impacts over time. In this case, once the reputation increases, the users requesting the service increase, and thus reputation exceeds such Cap_S value, users are dispersed and inDemand undergoes a faster decrease. In this figure,

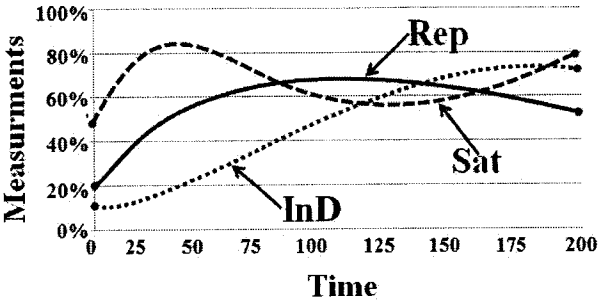


Figure A.3: Evolution of cooperative parameters for a community of Web services over time.

inDemand gets the highest value since a high reputation is followed by a large number of requests while satisfactions look more steady. In contrast, after a low quality service, all curves head down among which, inDemand decreases more since users stop requesting until such a single Web service manages to handle user requests again.

Figure A.3 illustrates the same structure analyzing the parameters regarding typical community C hosting some Web services. In this Figure, we see a more stable inDemand, which reflects C's stable market share. Once the curves tend to approach each other at the end. This extends to more details about parameters of a stable community that managed to maintain a tradeoff between its capacity and inDemand.

In this appendix, and based on the observations on the experimental result of the simulations, we can conclude that the Performance measurement by which a single Web service is encouraged to join a community within which a better handling ability over the users' requests is guaranteed. This empirical analysis takes into account the system parameters and motivates higher performance even under lower reputation level. In this analysis, single Web services are allowed to predict their further reputation level (and thus, performance) that let them make the best decision.