Dynamic Formation and Strategic Management of Web Services Communities

Ehsan Khosrowshahi-Asl

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy at

Concordia University

Montréal, Québec, Canada

July 2015

CONCORDIA UNIVERSITY

Division of Graduate Studies

This is to certify that the thesis prepared

By: **Ehsan Khosrowshahi-Asl**

Entitled: **Dynamic Formation and Strategic Management of Web Services Communities**

and submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________ Dr. Christian Moreau

______________________ Dr. Muhammad Younas

______________________ Dr. Peter Grogono

______________________ Dr. Ferhat Khendek

______________________ Dr. Joey Paquet

______________________ Dr. Jamal Bentahar

______________________ Dr. Hadi Otrok

Approved by ______________________________

Chair of the CSE Department

__________ 2015 ______________________________

Dean of Engineering

ABSTRACT

Dynamic Formation and Strategic Management of Web Services Communities

Ehsan Khosrowshahi-Asl, Ph.D.

Concordia University, 2015

In the last few years, communities of services have been studied in a certain numbers of proposals as virtual pockets of similar expertise. The motivation is to provide these services with high chance of discovery through better visibility, and to enhance their capabilities when it comes to provide requested functionalities. There are some proposed mechanisms and models on aggregating web services and making them cooperate within their communities. However, forming optimal and stable communities as coalitions to maximize individual and group efficiency and income for all the involved parties has not been addressed yet. Moreover, in the proposed frameworks of these communities, a common assumption is that residing services, which are supposed to be autonomous and intelligent, are competing over received requests. However, those services can also exhibit cooperative behaviors, for instance in terms of substituting each other. When competitive and cooperative behaviors and strategies are combined, autonomous services are said to be "coopetitive". Deciding to compete or cooperate inside communities is a problem yet to be investigated.

In this thesis, we first identify the problem of defining efficient algorithms for coalition formation mechanisms. We study the community formation problem in two different settings: 1) communities with centralized manager having complete information using cooperative game-theoretic techniques; and 2) communities with distributed decision making mechanisms having incomplete information using training methods. We propose mechanisms for community membership requests and selections of web services in the scenarios where there is interaction between one community and many web services and scenarios where web services can join multiple established communities. Then in order to address the coopetitive relation within communities of web services, we propose a decision making mechanism for our web services to efficiently choose competition or cooperation strategies to maximize their payoffs. We prove that the proposed decision mechanism is efficient and can be implemented in time linear in the length of the time period considered for the analysis and the number of services in the community. Moreover, we conduct extensive simulations, analyze various scenarios, and confirm the obtained theoretical results using parameters from a real web services dataset.

ACKNOWLEDGEMENTS

I would never have been able to finish my dissertation without the guidance of my committee members, help from friends, and support from my family.

I would like to thank my supervisors, Prof. Jamal Bentahar and Prof. Hadi Otrok for giving me the opportunity to work under their supervision. I am very grateful to them for their valuable suggestions and guidance throughout the preparation of this thesis. I learned a lot of valuable lessons which will be useful for me beyond the scope of this thesis throughout my lifetime. I also would like to thank Prof. Rabeb Mizouni for her significant help during my research work.

I would like to thank my examiner committee Professors Peter Grogono, Ferhat Khendek, Joey Paquet and Muhammad Younas for giving me the honor by being in my PhD committee. Their time and effort are greatly appreciated.

I would like to thank my colleague Babak Khosravifar who was always willing to help and give his best suggestions. Also, I would like to thank my friends and lab colleagues Omar Marey, Faisal Al-Saqqar and Khalid Sultan for their help and support.

Finally, I am very grateful to my fiance and colleague Atieh Saberi and my parents for their understanding, encouragement and their endless support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

AI	Artificial Intelligence
API	Application Program Interface
BSV	Bilateral Shapley Value
C	Community of Web Service
CFV	Community Feature Vector
CFVS	Community Feature Vector Set
CS	Coalition Structure
DDM	Distributed Decision Making
IT	Information Technology
NP	Nondeterministic Polynomial time
OSC	Open Software Connectivity
PO	Potential Output
QoS	Quality of Service
QWS	Quality of Web Service
REST	REpresentational State Transfer
ROC	Receiver Operating Characteristic
SOAP	Simple Object Access protocol
UDDI	Universal Description, Discovery and Integration
W3C	World Wide Web Consortium
WS	Web Service
WSDL	Web Service Definition Language
XML	EXtensible Markup Language

Chapter 1

Introduction

In this chapter, we introduce the context of this research, which is about communities of web services abstracted as autonomous agents. Those agent-based web services use intelligent decision making mechanisms to improve their performance in multi-agent setting. We discuss the motivations of this work and briefly review the literature to identify the problems we aim to solve in this thesis. Moreover, we discuss our objectives and contributions.

1.1 Context of Research

Over the past years, online services have become part of many scalable business applications. The increasing reliance on web-based applications has significantly influenced the way web services are engineered. Web services provide a set of online software functions accessible at a network address over the web. The recent developments are shifting web services from passive and individual components to autonomous and group-based components where interaction, composition, and cooperation are the key challenges [69, 16]. The main objective is to achieve a seamless integration of business processes, applications and web services. Delivering high quality services considering the dynamic and unpredictable nature of the Internet is still a very critical and challenging issue.

Typically, web services are business applications deployed as autonomous and interoperable agents [49]. In fact, the W3C consortium defines a web service as “an abstract notion that must be implemented by a concrete agent”. However, the web is stocked with agent-based services that offer similar business functionalities, which leads to service consumers having difficulties in choosing the most appropriate agents to interact with.

The need for highly available and responsive services has called for grouping and collaborative mechanisms of loosely-coupled web services, particularly in business settings. The idea of grouping web services within communities and the way those communities are engineered so that web services can better collaborate have been proposed and investigated in [53, 14, 71]. Communities are virtual groups of web services having similar functionalities [89, 66, 59, 52], but probably different non-functional quality attributes, which form the QoS parameters. Communities aim to provide higher service availability and performance than what individual web services can provide.

1.2 Motivating Scenario

In this section, we present a scenario and demonstrate why there is need for communities of services. We first propose an example of a real world scenario, focusing on user experience. There are a plethora of options available to people in today’s society, including weather forecasting, ticketing services, map services, local places guides and so on. Most mobile or web applications cannot independently satisfy users requests and should rely on different online services. The high competition within the services industry requires applications to use reliable and high quality online services.

If the user were to check a web site or run an application on her mobile device upon having downtime, or having high response delay or encountering any non-satisfying quality metric, she will instantly remove the application, which is a huge business concern for application providers. For example, if a user installs a ticketing application on her mobile device and the application is not using high quality service providers, the user would instantly uninstall the application, which has an extremely negative impact on the visibility of the application. Thus, end user satisfaction is the main goal for competitive online providers. Communities of web service, by providing services with higher quality, higher uptime and reliability for end users, aim to reach this goal. To this end, community management decisions should capitalize on important QoS parameters while forming the community and during membership management.

High demand on online services has created a massive business competition. For example, nowadays users are provided with multiple choices of web services offering local places information such as coffees, venues, and shops nearby a geographic position. It is hard for new web services to find their customers and be visible for end users amongst hundreds or thousands of other available services, even if they provide a high quality of service. Hence, the concept of communities of web services provides them with the chance of joining a platform with an established market share and reputation. However, it is crucial for a community manager to consider many factors when inviting or accepting new members. For example, if the market share is not big enough, bringing new web services can cause revenue drop for the already residing members. This may encourage other web services to collude, leave, or join other communities, hurting the community stability. This is an important issue which has not been addressed previously in the relevant literature. On the other hand, if communities bound the number of web services to ensure higher revenue, availability and response time could be negatively affected if some members encounter problems. This is because alternative web services for substitution will be limited. This has also not been efficiently addressed in the related work. Consequently, community formation algorithms satisfying some desirable properties such as community stability and overall revenue are yet to be defined considering end users, community managers and service providers.

1.3 Problem and Research Questions

Web services communities are dynamic by design [53]. In these communities, web services are modeled as intelligent autonomous agents, where they can adopt a strategy maximizing their payoff at any time. A web service can ask joining a community and has the right to leave it. Community managers can invite or ask a web service to leave in order to maximize the community profit. Users can simply stop sending requests to a web service which is not providing satisfactory services. Thus, it is important to consider all the parties involved in the decision making process about the community formation and management. Most of the recent work on communities of services are either user-centric and focus on user satisfaction [23] or system-centric and focus on the whole system throughput, performance and utilization. There are many contributions in distributed, grid, cluster and cloud services which are system-centric. However, in real world environments and applications, both users and service providers are self-interested agents, aiming to maximize their own profit. In those environments, both parties (users and services) will collaborate as long as they are getting more benefits and profit. Our initial research question is:

How can we model the community of agent-based services in order to maximize the profit of involved users, web services and community organizers? [R1]

In order to address this problem, recently [46, 41, 48] proposed mechanisms to help users and services maximize their gain. A two-player non-cooperative game between web services and community master (i.e., manager) was introduced in [41]. In [46], a 3-way satisfaction approach for selecting web services has been proposed. In this approach, the authors proposed a web service selection process that the community masters can use. The approach considers the efficiency of all the three involved parties, namely users, web services and communities. The issue with these solution concepts is that they consider community as a whole and model it as one entity in their formulations. A community master decides on behalf of all the members using an aggregated function of parameters. This can hurt the overall revenue for some individual web services, or even a subset of web services. Those services can collude and form their own community to increase their payoff, instead of having to adjust and share their resources with other members. Another important issue which needs to be considered is the community stability. In community of web services, the members and community organizers collaborate to perform tasks. Having jointly completed a task and generated revenue, they need to agree on some reasonable method of dividing profits (or tasks) among themselves. This is a key issue for the group stability still to be investigated. If the revenue sharing mechanism is not fair enough for any subset of web services working in the community, these agents, as profit maximizing entities, would deviate and make their own group. Thus, an important an important research question that we would like to address is:

How can we model fair and stable communities as coalitions of agent-based web services? [R2]

In [48], a cooperative scheme among autonomous web services based on coalitional game theory has been introduced. The authors have proposed an algorithm to reach individually stable coalition partition for web services in order to maximize their efficiency. The communities choose new web services on the promise that it would benefit the community without decreasing any other web service's income. In their model, the worth of community is evaluated with high emphasis on the availability metric and considering price and cost values only. The community structure is based on a coordination chain, where a web service is assigned as a *primary* web service and the community task distribution method will initially invoke the primary web service. Only if the primary web service is unavailable, the next backup web services in the ordered coordination chain will be invoked. However, in cooperative models, it is preferred to have a real and active cooperative activity engaging all agents to perform the tasks more efficiently. Thus, the final research question we aim to investigate is:

How can we model and analyze the cooperation among the community members in realistic, applicable and practical settings? [R3]

In most of the recent work on communities of web service, the solutions consider the architecture of centralized management for communities where most of the decisions are made by the centralized coordinator. However, in real world scenarios, decisions made by independent service providers are highly distributed.

How can we model a distributed decision making process for the problem of forming communities of services? [R4]

Also in all of the mentioned proposals, the community manager as a centralized entity, has complete information of all the web services and their quality. However, centrality and complete information are strong assumptions, which are not fully compatible with real business scenarios.

How can we make web services operate efficiently based on limited information? [R5]

Within communities, services can exhibit competitive behavior as they provide the same functionalities and the number of users requests is finite. However, for the same reason of being functionally similar, services can cooperate with each other, for example to substitute each other in order to perform some sub-tasks. For instance, services can opt for performing tasks if they feel they are capable enough or decide to cooperate by showing the availability to perform some sub-tasks. The relevant question to be addressed is:

How can we design a community model where both competitive and cooperative behaviors exist? [R6]

1.4 Contributions

1.4.1 Contribution 1: Efficient Coalition Formation for Web Services

In this research work[1], our first objective is to propose a cooperative model as game for the aggregation of web services within communities. The solution concepts of our cooperative game seeks to find efficient ways of forming coalitions (teams) of web services so that they can maximize their gain and payoff, and distribute the gain in a fair way among all the member services. Achieving fairness when the gain is distributed among the community members is the main factor to keep the coalition stable as no web service will expect to gain better by deviating from the community. In other words, the coalition is made efficient

[1]This contribution was published in [8]

if all the members are satisfied. We first propose a representation function for communities of web services based on their QoS attributes. By using this function, we can evaluate the *worth* of each community of web services. When facing new membership requests, a typical community master checks whether the new coalition having the old and new set of web services will keep the community stable or not. The community master will reject the membership requests if it finds out that the new coalition would be unstable, preventing *any* subset of web services from gaining significantly more by deviating from the community and joining other communities or forming new ones. The computation of solutions for cooperative games is combinatorial in nature and proven to be NP-complete [27], making this computation impractical in scalable real world applications. However, using the concepts of coalition stability, the second objective is to investigate approximation algorithms running in polynomial time providing web services and community masters with applicable and near-optimal decision making mechanisms.

1.4.2 Contribution 2: Distributed Decision Making for Dynamic Formation of Web Services Communities

In order to address the centralized decision making process limitations and adopt a distributed decision making approach, we introduce DDM[2], a Distributed Decision Making model for community formation. DDM regulates web service agents' decision making process in terms of cooperating and deciding which group to join and which service to invite for joining. Unlike existing work on community formation, our decision model is extracted from a data model in the form of information obtained from a large number of web services regarding their single and cooperative utilities as well as environmental parameters such as demand, service quality, etc. The generated decision tree improves agents' understanding

[2]This contribution is submitted to International Journal of Decision Support Systems

of the environment and and helps them select actions that lead towards maximizing their utilities. The advantage of this approach is that the tree, which is initially created from the past data, reflects a comprehensive vision about agents' attitudes in terms of their action selection based on their past experiences. Moreover, the tree is getting continuously updated based on both new received feedback and the outcome of chosen actions. This continuous update makes the approach adaptable to any change in the environment. The decision model provides web services with enough information which helps those services efficiently decide and predict the outcome of their different possible collaborations. This model works in a distributed manner in which services are self-sufficient in their decision making and do not rely on a centralized decision making process. Our findings show that communities of web services can efficiently find the appropriate web service to invite for cooperation as well as allowing a single web service to find the best communities to join. The proposed model can be seen as a recommneder system that suggests beneficial actions for both communities and single services. Communities can consider the decision model and analyze the characteristics of different individual web services and make prudent decisions when inviting a web service to join or accepting a join inquiry initiated from a web service. In general, DDM equips web services with efficient methods for foreseeing how their choices will impact both their short-term and long-term goals; therefore, opting for the best decision available.

To effectively generate the decision model for web services, we used a real dataset to extract web services' individual characteristics and used them to measure outcomes when these services cooperate with one another. The dataset has been extracted from real-world QoS evaluation results from 142 users on 4,532 Web services during 64 different time slots. Combining the available data based on each web service point of view on different time slots, we acquired 5 different unique features for those 4,532 web services. By engineering and extracting these features, we gathered functional and cooperative features for both individual web services and communities in different time slots. We were able to investigate the path a web service might take to achieve the best utility out of effective interactions with others. All the paths and outcomes are labeled to be utilized in the training model. Using cross validation sets, web services are able to compute the optimal hypothesis function (using logistic regression) that can be used to predict outcomes of cooperative work with other individual web services or communities. Our findings show that web services equipped with DDM have by far better outcomes than the ones that either do not cooperate or randomly find communities to join.

1.4.3 Contribution 3: Analyzing Coopetition Strategies of Services within Communities

In the previous contributions, our focus was on community formation and we emphasized cooperative behavior of the web services as agents. In our next contribution[3] our focus is on the internal management affairs of communities of web services. Within communities, the web services, selfish and utility maximizers by nature, can follow two different strategies, namely cooperation and competition in order to increase their payoffs when they provide services to consumers [50]. When competitive and cooperative behaviors and strategies are combined, autonomous services are said to be “coopetitive”. In typical business settings, services are used to compete within communities as they provide the same functionalities and the number of users requests is finite. However, the same reason of providing similar functionalities can lead services to cooperate because they can replace each other in case of failure or unavailability, and services can do better in a coalition structure. Analyzing

[3]This contribution was published in [7]

services competition and cooperation strategies within communities is still an open problem that motivates the research described in this section. we propose a mechanism within which service agents in the community could choose either to compete for an announced task[4], or to cooperate with other competing services in the same community to accomplish some subtasks of the announced task. We equip intelligent web services to follow a reasoning technique to choose best interactive strategy (Coopetitive attitude, which is categorized to compete and cooperate). In the proposed system, We explore details behind the strategic decision making procedures and enable service agents to apply different techniques to constrain high efficiency and obtain the maximum utility. We investigate services' expected payoffs and the involved probabilities that are used to choose over the two interacting strategies.

To summarize, the main problem we aim to tackle in this thesis is the formation of stable and efficient coalitions maximizing web services and community revenue. The main objectives are:

- To propose a cooperative model and analyze its solution concepts in order to address the problem of optimizing coalition formation for a stable community.
- To reduce the complexity of computing the solution concepts of the cooperative model tailored to the problem of communities of agent-based web services in order to make these solutions applicable in real world scenarios.
- To analyze the effect of different membership and taxation models that the master can apply to the members on the stability of the community.
- To investigate the impact of learning on individual and group decision making within the cooperative model of the community.

[4]Requests and tasks are used in this thesis interchangeably.

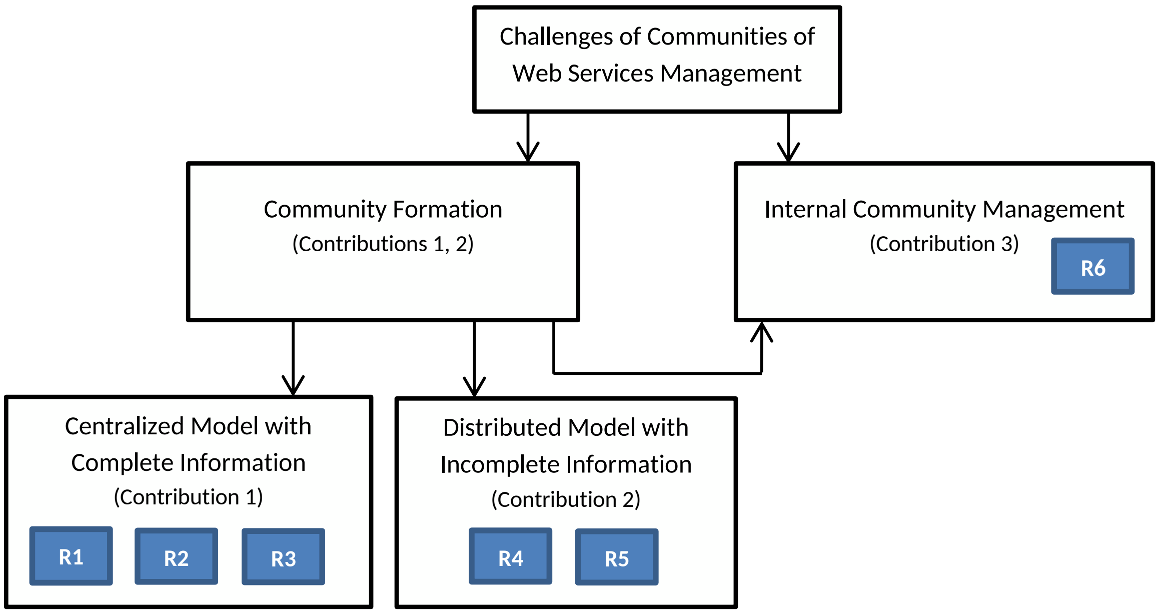

Challenges of Communities of Web Services Management
Community Formation
(Contributions 1, 2)
Internal Community Management
(Contribution 3)
R6
Centralized Model with Complete Information
(Contribution 1)
R1
R2
R3
Distributed Model with Incomplete Information
(Contribution 2)
R4
R5

Figure 1.1: The Proposed Framework.

- To design the community decision making process in a distributed manner and train agents so they can operate efficiently when information is incomplete.
- To validate the proposed methods by extensive simulations and comparison with other similar proposals.

Figure 1.1 highlights our contributions and proposed model for communities of web services formation and management.

1.5 Thesis Organization

The rest of the proposal is organized as follows: We present in Chapter 2 the background needed for our research along with relevant related work. We introduce the web services and the concept of communities of web services and the theoretical background used throughout the these. Chapter 3 provides an efficient method of coalition formation for web services. In this chapter we have addressed $[R1]$, $[R2]$, $[R3]$ research questions on efficient ways of community formation. Chapter 4 presents a distributed method of formation of web services communities. In this chapter we address [$R4$], [$R5$] research questions by proposing DDM, a distributed decision making mechanism for our web services which can perform in distributed manner and when information is incomplete. In Chapter 5 we delve into internal management of communities of web services and we address the [$R6$] research question by analyzing the cooperative behavior within the communities of web services. Finally in Chapter 6, we present our conclusion and future plan.

Chapter 2

Background

In this chapter, we briefly review web services, then we introduce the concept of communities of web services, their architecture and applications and the benefits of forming communities. Thereafter, we discuss the cooperative game theory concepts used throughout the thesis. Finally, we discuss relevant related work on web service communities and games in the literature of service oriented computing.

2.1 Community of Web Services

In this section, we present web services and discuss the concept of their communities from architectural and operations perspectives.

2.1.1 Web Services

Over the past years, online services have become part of standard daily life of people around the globe. Many modern applications rely on web services from different providers. For instance, many mobile and tablet applications which have limited storage and processing power are merely interfaces aggregating different information from online services. Examples are vast, weather forecasting, ticket selling, shopping apps, local maps and places searching are some of them.

The World Wide Web Consortium (W3C) defines web services as follows: “software system designed to support interpretable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with XML serialization in conjunction with other Web-related standards”. When developers declare a new web service, it will be discovered based on its description that fully discloses its functionalities. Developers also have to declare a public interface and a readable documentation to help other developers when integrating different services [22]. Nowadays, web API standards which do not require XML-based web service protocols like SOAP and WSDL are also emerging. They are also called REST (representational state transfer) services which are moving towards simpler communication protocols.

We are not going to delve into engineering details of online web service implementation and its protocols in this thesis. We are interested in web services from their business model perspective. Service providers usually charge end users for services they provide. For example, Google has listed their pricing and plans for wide range of services they provide on their web service console page[1].

In our research work, we abstract web services as rational agents[2] providing services to end users. They aim to maximize their individual income by receiving enough requests from end users. In order to increase their revenue, web services seek for more tasks if they have the capacity and throughput to do so. Web services can join communities to have

[1] https://code.google.com/apis/console

[2] The term rational is used here in the sense that web services are utility maximizers

better efficiency by collaborating with others, to have access to broad market share, and to have opportunity of receiving a bigger task pool from end users. Furthermore, the high reliance on web services has increased quality expectations from end users. Communities of web services can provide higher availability, performance, reliability, and recovery for end users.

2.1.2 Web Services Communities

Community refers to "the condition of sharing or having certain attitudes and interests in common" or "a group of people living in the same place or having a particular characteristic in common"[3]. In [14, 89], the authors introduce community of web services as collection of cooperative web services with common functionalitiers but different QoS metrics. Therefore *communities* are differentiated from *composition* types of web service cooperation in which web services with different functionalities work together to generate a new service with composite functionality.

Maamar et al. initially in [51] and then comprehensively in [53] proposed an architecture utilizing *Contract-Net* protocol for engineering task distribution within communities. This architecture has been further developed in [15, 43, 45, 54]. Two types of roles have been distinguished for community members: masters and slaves. Master web services lead communities and are responsible for membership management. They can invite and convince slave web services to join the community, and attract new slave web services to their communities by awarding them better payoff. Moreover, they can eject some slave members from the community to improve its overall reputation if these members are misbehaving or cannot provide the promised QoS [53].

Figure 1 depicts the basic architecture of communities of web services. The main

[3]Oxford Dictionaries

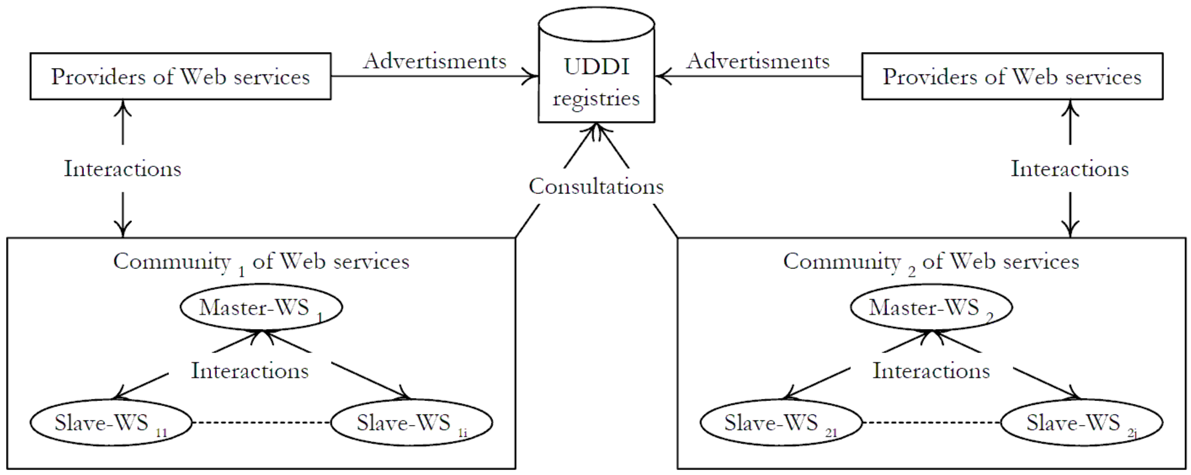


Figure 2.1: Communities of Web Services Architecture as Proposed in [53].

components of the architecture are: 1) the providers of web services; 2) UDDI registries; and 3) communities platform. Communities abstract the same model of defining, announcing and invoking web services. They also adopt the same protocols that standard web services use with UDDI registries. UDDI is a platform-independent XML based registry list which facilitates worldwide web service discovery.

2.2 Cooperative Game Theory and Multi-Agent Systems

The theory of cooperative games is a branch of game theory that is a branch of game theory that studies strategies of self-interested entities or agents in a setting where those agents can increase their payoff by binding agreements and cooperating in groups. We let N be a set of players which can form a group called a *coalition*. A *coalitional game* is a pair $G = (N, v)$, where v is called a *characteristic function* $v : 2^N \rightarrow \mathbb{R}$, mapping the set of players of the coalition to a real number $v(N)$, the worth of N. This number usually represents the output or payoff or again the performance of these players working together as coalition. If a coalition S is formed, then it can divide its worth, $v(N)$ in any possible way among its members. The payoff vector $x \in \mathbb{R}^N$ is the amount of payoff being distributed among the members of the coalition N. The payoff vector satisfies two conditions:

- $x_i \geq 0$ for all $i \in N$, and
- $\sum_{i \in N} x_i \leq v(N)$

The second criterion is called the *feasibility* condition, according to which, the payoff for each agent cannot be more than the coalition total gain. A payoff vector is also *efficient* if the payoff obtained by a coalition is distributed amongst the coalition members: $\sum_{i \in N} x_i = v(N)$. This definition of the characteristic function works in *transferable utility* (TU) settings, where utility (i.e., payoff) is transferable from one player to another, or in other words, players have common currency and a unit of income that is worth the same for all players [62].

When dealing with cooperative games, two issues need to be addressed:

1. Which coalitions among all possible coalitions to form?
2. How to reward each member when a task is completed?

The following sections help address these two issues.

2.2.1 Cooperative Game Concepts

Definition 1 (Shapley value) Given a cooperative game (N, v), the *Shapley value* of player i is given by [78]:

$$\phi_i(N,v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N|-|S|-1)!}{|N|!} (v(S \cup \{i\}) - v(S)) \tag{2.1}$$

Shapley value is a unique and fair solution concept for payoff distribution among the members of the coalition. It basically rewards members with the amount of marginal contribution they have to the coalition. It checks the contribution of member i by adding the agent, to all possible subsets of coalitions S, where $S \subseteq N \setminus \{i\}$. If he is added to the set S, his contribution to the coalition is $v(S \cup \{i\}) - v(S)$. Average marginal contribution of agent i's is calculated by averaging this value over all possible subsets of N, in *Shapley value* equation (2.1).

Definition 2 (Core) A payoff vector x is in the *core* of a coalitional game (N, v) if and only if[64]:

$$\forall S \subseteq N, \sum_{x_i \in S} x_i \geq v(S) \tag{2.2}$$

The core is basically a set of payoff vectors where no subset of players S' could gain more than their current payoff by deviating and making their own coalition $\sum_{i \in S'} x_i \geq v(S')$. The sum of payoffs of the players in any sub-coalition S is at least as large as the amount that these players could earn by forming a coalition by their own. In a sense, it is analogue to Nash equilibrium, except that core is about deviations from groups of entities. The core is the strongest and most popular solution concept in cooperative game theory. However, its computation is a combinatorial problem and becomes intractable as the number of players increases. The core of some real-world problem games may be empty, which means having the characteristic function of the game (N, v), there might be no possible distribution of payoff assuring stability of subgroups.

Definition 3 (Convex cooperative games) A game (N, v) with characteristic function $v(S)$ is convex if:

$$v(S) + v(T) \leq v(S \cup T) + v(S \cap T), \forall S, T \subseteq N. \tag{2.3}$$

According to a classic result by Shapley [77], convex games always have a non-empty core. We will use a variation of convexity condition in our algorithm to check whether our coalitions are stable.

ε-core

When the *core* set of a game is empty, it means no coalition of players can gain anything by deviating. An outcome would be unstable if a coalition can benefit even by a small amount from deviating, which is a strong requirement. In fact, in some situations, deviations can be costly, or players may have loyalty to their coalitions, or even it can be computationally intractable to find those small benefits. It would only make sense for a coalition to deviate if the gain from a deviation exceeds the cost of performing the deviation. ε-*core*[79] relaxes the notion of the core, and only requires that no coalition would benefit significantly, or within a constant amount(ε) by deviating (see Equation 2.4).

$$\forall S \subseteq N, \sum_{x_i \in S} x_i \geq v(S) - \varepsilon \tag{2.4}$$

Coalition Structure Formation

Coalition structure formation is the problem of finding the best partition of web services into teams. In these settings, the performance of an individual service is less important than the *social welfare* of the whole system, which is the sum of the values of all teams. Having the game (N, v), a coalition structure (CS) is *socially optimal* if CS belongs to set $\arg\max_{CS} v(CS)$ where $v(CS)$ is the sum of the values of all coalitions inside CS. $v(CS) = \sum_{C \in CS} v(C)$.

Example 1. Consider a game $G = (N, v)$, with two players where $N = 1, 2$. Each of these players can produce 5 units of output working alone and by collaborating they can produce 20 units worth of output. Therefore we have: $v(1) = 5, v(2) = 5, v(1, 2) = 20$. The *core* of the game, which is the set of all possible distribution of gain among players guaranteeing

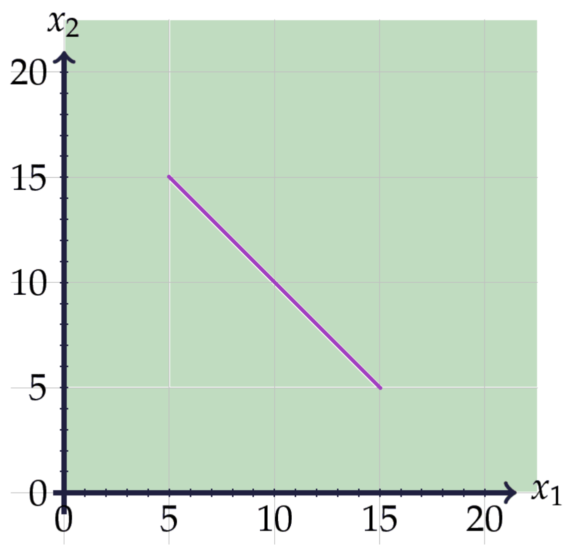


Figure 2.2: Core of the 2-player game of example 1.

stability is: $core(N,v) = \{(x_1,x_2) \in R^2 | x_1 >= 5, x_2 >= 5, x_1 + x_2 = 20\}$, as illustrated in Figure 2. Distributing the 20 units of income, among these two players, for all the points in the line will make outcome stable, since none of these players can gain more than 5 by working alone. However although they have same qualities, the core can suggest a stable outcome where one agent can earn three times more than the other agent: $\{5,15\}$. As mentioned in previous section, *core* result may not be fair, the *core* only considers stability. However, *Shapley value* considers fairness. According to Equation 2.1, the two workers should each share 10 units of income, since they have the same marginal contribution to all subsets of the coalition. As you can see the distribution vector of $\{10,10\}$ is also a member in *core* set. Later we are going to show if *core* of a coalition game is not empty, and game is convex, the shapley value lies within *core* set.

Example 2. In this example, we want to analyse games under conditions which core can be empty. Consider a game $G = (N,v)$, where $N = 1,2,3$ and $v(\{i\}) = 0$, $v(\{Ci\}) = \alpha for |C| = 2$ and $v(\{N\}) = 1$. The (x_1,x_2,x_3) distribution vector according to Equation 2.2, is in *core* if $x_i \geq 0$ which implies each player will get more than 0 which they would when working alone and $\forall i, \forall j, x_i + x_j \geq \alpha$ which implied any pair of players will get more than α which they would earn if they worked in pair without the third player and finally $\sum_{i \in N} x_i = 1$ which implied all the gain is distributed among the three players. Based on these three equations we have, $\forall i \in N 0 \leqslant 1 - \alpha$ and $\sum_{i \in N} x_i = 1$. By summing first equation for all three platers, we conclude $Core(N, v)$ is nonempty iff $\alpha \leqslant \frac{2}{3}$. When alpha is more than $\frac{2}{3}$, the contribution of third player is not good enough to justify the group of three players working together. The third player will increase the revenue with less then $\frac{1}{3}$, and the other two players, both can get better share of revenue if they work together. This is why the group of three players working together when $\alpha > \frac{2}{3}$ is not stable.

2.2.2 Stability of Coalitions

Core stability is a highly desirable property. However, in many problems this property is not achievable. It would be more ideal to maintain a set of quasi-stable payoffs when the core is empty. There are several approaches to achieve this goal. One may drop the stability requirement and focus on other types of solutions for which a payoff division is guaranteed to exist. Two well known solution concepts in this category are *nucleolus* [76] and the *bargaining set* [25]. They try to minimize some measure of unhappiness in the game for the agents.

Another approach to stabilize the game can be achieved via external subsidies. When *Core* is empty it means the game is not stable since the coalition is unable to generate enough revenue to satisfy the demands of each subset of agents. An external party that is interested in stabilizing the game provides a subsidy to the agents if they form the grand coalition, and thus a value of $\lambda v(C)$ is divided among them, where $\lambda \geq 1$. Clearly any game can be stabilized using a large enough λ, however the external party would be interested in the minimal subsidy required in order to stabilize the game.

A community can also be stabilized by relaxing the core constraints. According to *Core*, an outcome is unstable if a coalition can benefit even by a small amount from deviating, which is a strong requirement. In fact, in some situations, deviations can be costly, or players may have loyalty to their coalitions, or even it can be computationally intractable to find those small benefits. It would only make sense for a coalition to deviate if the gain from a deviation exceeds the cost of performing the deviation. *ε-core* relaxes the notion of the core, and only requires that no coalition would benefit significantly, or within a constant amount (ε) by deviating (see Equation 2.5).

$$\forall S \subseteq N, \sum_{x_i \in S} x_i \geq v(S) - \varepsilon \tag{2.5}$$

Alternatively, ε can be thought of as a *tax* imposed on a coalition should it choose to deviate. This can again be seen as an external party, trying to stabilize the coalition by imposing some *tax* on deviation. Taxation and subsidizing as methods of stabilizing cooperative games have been studied in [13, 11, 60].

2.2.3 Representation and Complexity Issues

Shapley value is the unique “fair” way to distribute the total surplus generated by the coalition, among all the players. The nature of the Shapley value is combinatorial, as all possible orderings to form a coalition needs to be considered. This computational complexity can sometimes be an advantage as agents cannot benefit from manipulation. For example, it is NP-complete to determine whether for a bunch of agents to collude and make their own coalition and guarantee an increase in payoff of all participants [88]. There are some representations that allow us to compute the Shapley value efficiently by reducing the input size of the problem. One example is *Induced subgraph games* which was introduced by Deng and Papadimitriou [28]. In this representation, players are represented by graph nodes, and their valuation function should be the sum of weights of all edges between the node and all its neighbors. It is a succinct representation, using an adjacency matrix, which needs only $O(n^2)$ space to store all the input, which is a major improvement from $O(2^n)$ because if weights of all the edges in graph are all positive, the Shapley value can be computed in time $O(n^2)$. However, this representation is not complete, some games cannot be represented by a induced subgraph game [88].

Ketchpel introduces the Bilateral Shapley Value (BSV) [39] for coalition games with general valaution functions. It reduces the combinatorial complexity of the computation of the Shapley value, breaking the community to multiple disjoint set. With backtracking and dynamic programming like methods, they merge and store the marginal contribution of disjoint coalitions, reducing the overall complexity of the algorithm. However, the solution is still NP-Complete and BSV time and space complexity grows exponentially.

In order to make cooperative game concepts practical in real world application, we have proposed an approximation multi-layer algorithm useful for service orinted computing settings. Our excrements illustrate, these algorithms can provide applicable and near optimal solutions for real world applications.

2.3 Related Work

2.3.1 Communities of Web Services

Here we introduce the related research work regarding the engineering and formation of communities of web services. In [14], Benatallah et al. defined communities as *Service Containers* that aggregate substitutable web services providing a common functionality (same set of operations). They abstracted *Service Containers* as web services that are created, advertised, discovered and invoked just as elementary web services. The *Container* is considered as a manager that is responsible for web service selection upon receiving a request on run-time. The authors have proposed a scoring service based on non-functional requirements of the request and web service capabilities to dynamically chose the web service to perform the requested task. A similar concept was proposed by Maamar et al. in [53]. The authors introduced web services communities as a collection of web services with a common functionality but different QoS properties. A community manager, upon receiving a request, delegates the request to one of its current members. The choice is based on the performance history and quality metrics of each web service. The authors have proposed an efficient global web service selection algorithm in order to approach quality constraints and preferences for composite services which require aggregation of different types of services to satisfy the user.

Benslimane et al. [17] have proposed a multi-layer approach grouping similar Web services into communities and having an interface implemented as an abstract web service for accessing the community on top of the community layer. The interactions between composite, management and community layers and the bindings are performed by a generic driver called Open Software Connectivity (OSC).

In [47], Limam and Akaichi have proposed web service communities with centralized access across distributed web services. They have proposed a framework for web service management, query resolution among communities and a query caching mechanism executed by the manager to improve the performance of query resolution process among many distributed communities. The key idea is to cache previous computed results for answering future queries.

Maamar et al. initially in [51] and then comprehensively in [53] proposed an architecture utilizing *Contract-Net* protocol for engineering task distribution within communities of web services. The protocol is centrally executed by the community manager. This architecture has been further developed in [54, 15, 43, 45]. Two types of roles have been distinguished for community members: masters and slaves. Master web services and community managers that lead communities and are responsible for membership management. They can invite and convince slave web services to join the community, and attract new slave web services to their communities by awarding them better payoff. Moreover, they can eject some slave members from the community to improve its overall reputation if these members are misbehaving or cannot provide the promised QoS.

In [59], Medjahed and Bouguettaya have developed a community as a “cluster” that groups Web services based on a specific area of interest. All web services in a given community share the same functionality. These communities are created by *third party community providers* which use the *community ontology* as a template and define a set of operations that all web services within a community should provide. Using semantic analysis on web service operations, web services either find and join a community with similar functionality or create a new operation description for a new community. The authors have described the concept of *community agents* associated to *community providers*. A community agent is responsible, among other things, of the registration of services with the community. An example of a community that provides health care services to senior citizens has been used. In this example, a governmental entity is needed to check the health care standards used by the members before authorizing them to be part of the community. Such a central entity is represented by the community agent. Thus, community agents are playing the role of community managers. In a close work [89], Zeng et al. have described a global planning selection algorithm and a delegation algorithm to be run when a request to execute an operation is received by the community. This needs a central entity to run those algorithms. Such entity plays the same role as the community coordinator or manager.

2.3.2 Web Services Community Formation

Most of the recent work on communities of services are either user-centric and focus on user satisfaction [23] or system-centric and focus on the whole system throughput, performance and utilization. There are many contributions in distributed, grid, cluster and cloud services which are system-centric. However, in real world environments and applications, both users and service providers are self-interested agents, aiming to maximize their own profit. In those environments, both parties (users and services) will collaborate as long as they are getting more benefits and payoff.

In this direction, recently [46, 41, 48] proposed mechanisms to help users and services maximize their gain. A two-player non-cooperative game between web services and community master was introduced in [41]. In this game-theoretic model, the strategies available to a web service when facing a new community are requesting to join the community, accepting the master's invitation to join the community, or refusing the invitation to join. The set of strategies for communities are inviting the web service or refusing the web service's join request. Based on their capacity, market share and reputation, the two players have different sets of utilities over the strategy profiles of the game. The main limits of this game model are: 1) its consideration of only three quality parameters, while the other factors are simply ignored; and 2) the non-consideration of the web services already residing within the community. The game is only between the community master and the new web service, and the inputs from all the other members and their influence on the master's decision are simply ignored. The consideration of those inputs and this influence factor is a significant issue as existing web services can lose utility or payoff because of the new member, which can result in an unhealthy and unstable group. The problem comes from the fact that the existing members should collaborate with the new web services, so probably their performance as a group can suffer. Existing members may even deviate and try to join other communities if they are unsatisfied. Those considerations of forming stable and efficient coalitions are the main contributions of our research work.

In [46], a 3-way satisfaction approach for selecting web services has been proposed. In this approach, the authors proposed a web service selection process that the community masters can use. The approach considers the efficiency of all the three involved parties, namely users, web services and communities. In this work, it is shown how the gains of these parties are coupled together using a linear optimization process. However, the optimization problem in this solution tends to optimize some parameters considering all web services regardless of their efficiency and contribution to the community's welfare. Moreover, there are no clear thresholds for accepting or rejecting new web services. The solution of the optimization problem could, for instance, suggest web services already residing within the community to increase or decrease their capacity to cover up the weakness of other parties in the system. However, a high performing web service could deviate anytime it finds itself unsatisfied within the community instead of adjusting its service parameters.

In [48], a cooperative scheme among autonomous web services based on coalitional game theory has been introduced. The authors have proposed an interesting algorithm to reach individually stable coalition partition for web services in order to maximize their efficiency. The communities choose new web services on the promise that it would benefit the community without decreasing any other web service's income. In the proposed model, the worth of community is evaluated with high emphasis on the availability metric and considering price and cost values only. The community structure is based on a coordination chain, where a web service is considered as a *primary* web service and the community task-distribution method initially invokes the primary web service and only if the primary web service is unavailable, the method invokes the next backup web services as they are ordered in the coordination chain. We believe that this coordination chain limits the cooperation power as it introduces a sort of hierarchy. However, in pure and open cooperative models, such as the one we propose in this thesis, active cooperation activities engaging simultaneously many agents so that they can perform the tasks more efficiently are being used. Moreover, if the availability is high, which is the case nowadays with the recent advancements in cloud and hardware infrastructures, the backup web services will end-up having a very low chance of getting jobs, especially the ones further in the chain. This will results in a considerable waste of web services capabilities.

2.3.3 Coopetitive Behavior Within Communities of Web Services

At the best of our knowledge, there is no work in the literature of service and agent computing addressing the issue of coopetition strategies and when to cooperate or to compete. However, some relevant proposals to our proposed model are the ones that address service selection and task allocation mechanisms. In many frameworks proposed in the literature, service selection and task allocation are regulated based on the reputation parameter [18, 72, 74, 85]. In [35], the authors propose a framework to match potential benefits of services while cooperating with one another. The interesting idea is to consider the benefits under four categories: innovation and learning, internal business process, customer, and financial benefits. Innovation and learning perspective focuses on the knowledge, skills, and systems needed to improve the business continually. Necessary factors to build strategic capabilities and efficiency in addressed in internal business process. Values that customers seek are considered in *customer perspective* and financial performance to maximize the shareholder value are analyzed in *financial perspective*. Their goal is to design the framework for cooperating web services, inline with business strategy of firms in IT industry. In [3], the authors present a dependable framework for cooperative service agents that is based on the tuple space coordination model. The intrusion-tolerant perspective is emphasized in the paper where several security mechanisms are developed to enable a reliable coordination system. The proposed frameworks mostly aim to facilitate the coordination mechanism between services. However, the opposite strategy of competing is not analyzed where services might be more successful when competing within the same group. In fact, services are not always willing to cooperate even if they have some common goals, particularly when they operate within groups such as communities. In such a context, service agents can follow different interacting strategies and have to decide when to compete and when to cooperate so that their ultimate goal, maximizing their incomes, can be better achieved. In our framework, we analyze those different strategies to help services in their decision making process when these agents function within communities. We enable service agents to reasonably evaluate and decide over their coopetition strategies, which means deciding when to compete and when to cooperate.

Furthermore, there are a number of related proposals that take into account the correlation between (web) services and the ways these services coordinate their actions to accomplish the required tasks. In [38, 37, 56, 58, 85], the authors propose to rank services based on their reputation in the system and to use this ranking as a means to facilitate cooperation of services. In those models, services rely on one another on the basis of the reputation ranking system, using, among other parameters, the QoS [81]. There are other models that facilitate cooperation mechanisms among services using various techniques. Examples of those techniques include 1) coordination between two types of behaviors associated with component services: operational and control behaviors [86]; 2) Services-based workflows

[83]; 3) transaction-based approaches [32, 70]; 4) agent coordination mechanisms [21, 34]; 5) logical techniques [63, 80]; and 6) community models, which are virtual structures that aim at increasing the visibility of services and facilitating their discovery and composition by hosting and gathering services having similar or complementary functionalities but different QoS parameters [44]. However, deciding about which strategy to choose when services are competing but still need to cooperate to accomplish complex tasks has not been addressed and kept as open issue in all these proposals as faithfully argued in [55, 40].

2.4 Conclusive Remarks

In this thesis, as the first contribution, we will tackle the issue of community formation in an efficient way for all the web services and communities involved. We will use game theory to propose a cooperative game model for the aggregation of web services within communities. The solution concepts of our cooperative game seeks to find efficient ways of forming coalitions (teams) of web services so that they can maximize their gain and payoff, and distribute the gain in a fair way among all the web services. Achieving fairness when the gain is distributed among the community members is the main factor to keep the coalition stable as no web service will expect to gain better by deviating from the community. In other words, the coalition is made efficient if all the members are satisfied. We first propose a representation function for communities of web services based on their QoS attributes. By using this function, we can evaluate the *worth* of each community of web services. When facing new membership requests, a typical community master checks whether the new coalition having the old and new set of web services will keep the community stable or not. The community master will reject the membership requests if it finds out that the new coalition would be unstable, preventing *any* subset of web services from gaining significantly more by deviating from the community and joining other communities or forming new ones. The computation of solutions for cooperative game theory problems is combinatorial in nature and proven to be NP-complete [27], making this computation impractical in real world applications. However, using the concepts of coalition stability, we proposed approximation algorithms running in polynomial time providing web services and community masters with applicable and near-optimal decision making mechanisms.

Next, we will tackle the issue of distributed model of web services and propose a decision model for scenarios where information is incomplete. We will propose a training model for the problem of membership management of communities of web services. Using the traning model we aim to create a decision making profile for each community and web service involved which provides them with a set of feasible and utility increasing moves. This will equip our web services with efficient methods of foreseeing how their choices of actions would impact their long-term and short-term goals, therefore they opted for best decision available. The ultimate goal is to choose the best decision when it comes to communities formation, among many possible short-term rational and utility increasing choices.

In our last contribution, the focus is on internal community management. We will introduce a game-theoretic based model to analyze the efficiency characteristics for the active services in open networks. The proposed framework will consider the chances of web services in joining a community in different cases with truthful and lying information service agents. The proposed game will analyze the existing Nash equilibrium and situations where the maximum payoff is obtained.

Chapter 3

Coalition Formation for Autonomous Web Services

In this chapter, we present our coalition model of agent-based web services within communities [9]. We start by describing the general architecture and considered parameters for web services. Thereafter, problem modeling and formulation will be introduced in terms of task distribution and community revenue. Web service cooperative games in different settings will follow along with simulation results.

3.1 Preliminaries

In this section, we discuss the parameters and preliminary concepts that we use in the rest of the chapter.

3.1.1 Architecture

Our system consists of three main types of entities working together:

1) Web services are rational entities that aim to maximize their utilities by providing high quality services to end users. They aim to maximize their individual income by receiving enough requests from end users. In order to increase their revenue, web services seek for more tasks if they have the capacity and throughput to do so. Web services can join communities to have better efficiency by collaborating with others, to have access to higher market share, and to have opportunity of receiving a bigger task pool from end users. Throughout this thesis, in our equations, we refer to web services as ws and to the set of web services hosted by a given community as C. To simplify the notation, sometimes we simply write ws instead of $ws \in C$ to go through the elements ws of the set C.

2) Master Web Services or the community coordinators, are representatives of the communities of web services and responsible for their management. Communities receive requests from users and aim to host a healthy set of web services to perform the required tasks. They seek to maximize user satisfaction by having tasks accomplished according to the desired QoS. In fact, higher user satisfaction will bring more user requests and increase the market share and revenue of the community.

3) Users generate requests and try to find the best available services. User satisfaction is abstracted as function of quantity and quality of tasks accomplished by a given service. Higher user satisfaction leads to higher trust of the community by users hence directing more requests towards that service provider.

3.1.2 Web Service Parameters

Web services come with different quality of service parameters. These parameters with a short description are listed in Table 3.1.

We adopted a real world dataset [2] which has aggregated and normalized each of these parameters to a real value between 0 and 1. Since requests are not shared among web services and are distributed among all of them inside a community, each one of them comes

Table 3.1: List of web service QoS parameters.

Parameter	**Definition**
Availability	Probability of being available during a time frame
Reliability	Probability of successfully handling requests during a timeframe
Successability	Rate of successfully handled requests
Throughput	Average rate of handling requests
Latency	The average latency of services
Capacity	Amount of resources available
Cost	Mean service fee
Regulatory	Compliance with standards, law and rules
Security	Quality of confidentiality and non-repudiation

with a given QoS denoted by (QoS_{ws}). We assume that (QoS_{ws}) is obtained by a certain aggregation function of the parameters considered in Table 3.1. We use this quality output later in evaluating the community *worth* or *payoff* function.

3.1.3 Web Services Communities

Figure 3.1 represents our revised architecture of web service communities where tasks are to be distributed among the members that are interested in forming stable coalitions. As discussed in Chapter 2, communities are essentially virtual platforms aggregating web services having similar and complementary functionalities and communicate with other entities such as UDDI registries and users using particular protocols. Web services join communities to increase their utility by having larger market share and task pool. Community coordinators or master web services are responsible for community development, managing membership requests from web services and distributing user tasks among the community members. Community coordinators try to attract quality web services and keep the community as stable and productive as possible to gain better reputation and user satisfaction, which results in having higher revenue.

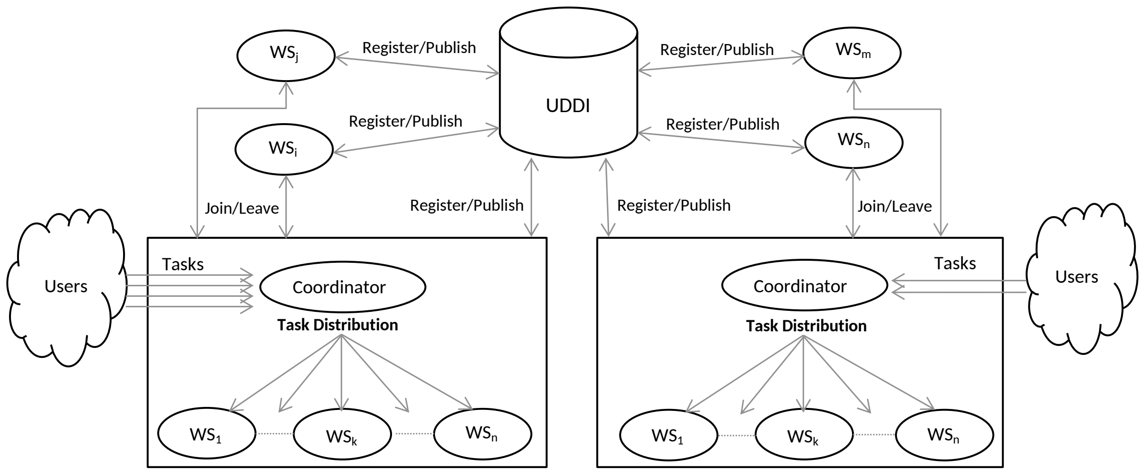


Figure 3.1: Architecture of Web Services communities.

3.2 Problem Formulation and Modeling

In this section, we present web services and community coordinator's interactions, the task distribution process and revenue models in web services communities.

3.2.1 Task Distribution

As mentioned in Section 3.1, communities are robust service providers with well established market share and reputation. By maintaining their reputation and performance, they attract end users which choose them as service providers to perform their tasks. The community master is characterized by a request rate (R_C) from users. Each web service comes with a given QoS (QoS_{ws}) from which the throughput Th_{ws} is excluded. Throughput is the average rate of tasks a web service can perform per time unit. Its exclusion from QoS_{ws} allows us to build our analysis on the particular value of Th_{ws}. Thus, web services perform tasks with an average output quality of QoS_{ws} and a throughput rate of Th_{ws}.

The community master uses a slightly modified *weighted fair queuing* method to distribute tasks among its members. The goal is to allocate incoming tasks to web services with a rate matching the throughput value of Th_{ws}. In *weighted fair queuing* method *all* the input flow is multiplexed along different paths, however in our case if the input rate (R_C) of the community is more than the summation of throughput values of the web services in the community, some of the input tasks will be queued and served with delay. Thus, the amount of tasks performed by community is $\sum_{ws \in C} (Th_{ws})$ when $\sum_{ws} Th_{ws} \leq R_C$. However, when the input rate (R_C) of the community is less than the summation of throughput values of the web services in the community, (R_C) the *weighted fair queuing* algorithm assigns a weighted task rate of $R_C \times \frac{Th_{ws}}{\sum_{ws} Th_{ws}}$ for each web service (ws) and the total rate of tasks being performed is R_C, the community's receiving request rate.

While distributing tasks, the community master can verify the performance, throughput and quality of service of tasks being performed by web services. It can recognize if web services are capable of doing the amount of tasks they advertised. If for any reason there is a decline in quality metrics or throughput, the community master will announce the new parameters and community masters and members can consider those values as benchmark for future performance calculations. Web services that got their quality declined are penalized, and in this way, players have incentive to reveal their real capabilities to profit best from the community and to avoid being penalized. In addition, the system should be dynamic enough to detect and react to web services quality metrics variation as over time web service metrics may degrade or improve, a change that the community should adjust to.

3.2.2 Community Revenue

The communities and web services earn revenue by performing tasks. The total gain is function of quality (QoS_{ws}) and throughput (Th_{ws}) of tasks being performed. We have adopted a linear equal weight average over the QoS parameters excluding the $Throughput$ and $Cost$ parameters. A community has the option to weigh specific QoS parameters depending on

Table 3.2: Case Study: Example 1

WS	QoS_{ws}	Th_{ws}	$Th_{ws} \times QoS_{ws}$
1	0.8	4	3.2
2	0.8	5	4.0
3	0.8	3	2.4

the expectations of their clients.

The maximum potential output of a community $(PO(C))$ is an aggregation of number of tasks, times their quality, for each web service member of the community:

$$PO(C) = \sum_{ws \in C} (T_{ws} \times QoS_{ws}) \tag{3.1}$$

If the summation of throughput values (Th_{ws}) of community members exceeds the input task rate of the community (R_C) the community cannot perform at its maximum potential. It denotes the case when the community has more web services than it needs to perform the input task load. The actual output has to be normalized to the amount of tasks being performed.

$$Out(C) = \begin{cases} PO(C) & \text{if } \sum_{ws} Th_{ws} \leq R_C \\ PO(C) \times \frac{R_C}{\sum_{ws} Th_{ws}} & \text{if } \sum_{ws} Th_{ws} > R_C \end{cases} \tag{3.2}$$

The revenue function of the web services community is a linear function of $Out(C)$ with a positive constant multiplier.

3.2.3 Case Study

In this section, we analyze three numerical examples and discuss the motivation of web services and community interactions and the strategies they can adopt and the revenue they can earn adopting these different strategies.

Community	Worth	Community	Worth
$\{1\}$	3.2	$\{1,2\}$	7.2
$\{2\}$	4.0	$\{1,3\}$	5.6
$\{3\}$	2.4	$\{2,3\}$	6.4
$\{1,2,3\}$	8.0		
Community R_C: 10			

Table 3.3: Case Study: Example 2

WS	QoS_{ws}	Th_{ws}	$Th_{ws} \times QoS_{ws}$
1	0.8	5	4.0
2	0.7	6	4.2
3	0.7	4	2.8

In the first example, we present the case of a community with $R_C = 10$, and three web services, each having different QoS_{ws} and Th_{ws} values as listed in Table 3.2. The worth of a community is calculated based on $Out(C)$ Equation (3.2) which is the amount of output being generated by the community. The first table lists the web services with their aggregated QoS_{ws} parameters, their task input rate while working alone, and also their throughput value Th_{ws}. The second table shows all the possible communities and their respective worth. The obtained values suggest that communities having more web services have better gain and output. However each community needs to distribute the gain between web services. Sometimes it is impossible to share the gain between all web services in a way that no subset of them would individually gain more if they form their own group. In this example, the value community of ws_1 and ws_2 is 7.2, With ws_3 joining the community the worth increases to 8.0. However there is no way to distribute the value among web services to have ws_1 and ws_2 earning 7.2, and ws_3 earning at least 2.4, the gain they could earn before joining the community. This fact makes the group unstable. In the second example, shown in Table 3.3, we even have situations where a web service (ws_3) joining a community

Community	Worth	Community	Worth
$\{1\}$	4.0	$\{1,2\}$	7.4
$\{2\}$	4.2	$\{1,3\}$	6.8
$\{3\}$	2.8	$\{2,3\}$	7.0
$\{1,2,3\}$	7.3		
Community R_C: 10			

Table 3.4: Case Study: Example 3

WS	QoS_{ws}	Th_{ws}	*Input Task Rate*
1	0.8	10	5
2	0.8	20	5
3	0.8	30	5

($\{ws_1, ws_2\}$) decreases the value of community. The reason is, the community is already full and all tasks are almost being distributed and new community with bad quality can degrade the average quality of tasks being done by the community. In both examples, the request of joining of web service ws_3 should be rejected by the community.

In Example 3, we consider the case of having different communities with different market share, R_C values. Web services also have a small share of market independently, providing them with a small task pull. In these kind of scenarios, the solution considers individual maximization of payoff and also the total worth of all communities which represents the *social welfare*. In this example the most efficient partition of web services is earned by having two coalitions of $\{C_{master_1}, ws_2\}$ and $\{C_{master_2}, ws_1, ws_3\}$, which yields a total value of $32 + 16 = 48$. In these types of scenarios, the goal is to reach stability, adopting a distributed approach where all players have the power of choice on the decision of whether or not they join a coalition. The communities usually start the game having some established members, encountering new web services, the communities may exchange web services and new web services would join them having at least one player gaining utility,

Community	Worth	Community	Worth
$\{C_{ms_1}\}$	0	$\{C_{ms_2}\}$	0
$\{C_{ms_1}, ws_1\}$	8	$\{C_{ms_2}, ws_1\}$	8
$\{C_{ms_1}, ws_2\}$	16	$\{C_{ms_2}, ws_2\}$	16
$\{C_{ms_1}, ws_3\}$	16	$\{C_{ms_2}, ws_3\}$	24
$\{C_{ms_1}, ws_1, ws_2\}$	16	$\{C_{ms_2}, ws_1, ws_2\}$	24
$\{C_{ms_1}, ws_1, ws_3\}$	16	$\{C_{ms_2}, ws_1, ws_3\}$	32
$\{C_{ms_1}, ws_2, ws_3\}$	16	$\{C_{ms_2}, ws_2, ws_3\}$	32
$\{C_{ms_1}, ws_1, ws_2, ws_3\}$	16	$\{C_{ms_2}, ws_1, ws_2, ws_3\}$	32
$\{C_{ms_1}, C_{ms_2}, ...\}$	0	$\{ws_1\}$	6.8
$\{ws_2\}$	4.2	$\{ws_3\}$	6.8
Community R_{C_1}: 20			
Community R_{C_2}: 40			

without hurting any other participant. In this example if we initially having two coalitions of $\{C_{master_1}, ws_2\}$ and $\{C_{master_2}, ws_1\}$ and a ws_3 as new web service, ws_3 joining C_{master_1} would hurt at least itself or ws_2, however ws_3 joining C_{master_2} would not hurt any participants and ws_3 would earn more within the community and the community will have enough web services performing the incoming tasks from users.

The first two examples illustrate the fact that a community cannot simply increase its revenue by adding more web services. The web services and even community owners are autonomous agents and would deviate and be displeased about the community if new members cause a drop in their profit. The job of the community master is to attract as many quality web services it can and keep them satisfied; hence the group stability is guaranteed. The third example highlights another type of problem we would like to address, which is how to form best possible groups of communities, and allocate web services among communities in a way which would maximize payoff for of our agents and members already residing in the communities. In next section, we provide collaborative game theory based algorithms for our autonomous agents, to tackle these problems and find applicable and efficient strategies for communities and web services to maximize their profit.

3.3 Web Service Cooperative Games

In this section, we present different web services community models and focus on the problem of how both web services and community masters as rational entities would adopt strategies to maximize their payoff.

3.3.1 Web Services and One Community

In this scenario, we assume the existence of a typical community managed by its master, and web services need to join it to be able to get requests from the master. The community master is characterized by a requests rate (R_C) from users. Each web service comes with a given QoS (QoS_{ws}). The worth of a community $v(C)$ is set to Out(C) based on Equation 3.2.

As mentioned in previous section, the worth and output of a community is a function of the throughput and provided QoS of its web service members. If the throughput rate is more than the master's input request rate, it means the web services inside the community are capable of serving more requests than the demand. Considering this factor, the valuation function is designed to balance the output performance so that it matches the exact throughput rate and QoS the web service can provide within the particular community.

In this first scenario, we only consider one grand coalition and analyze the system from the point of view of one single master web service and a collection of web services. The master web service decides which members can join the community and distributes the requests and income among its community members (see Figure 3.2).

The membership decision is made based on throughput and QoS of the considered

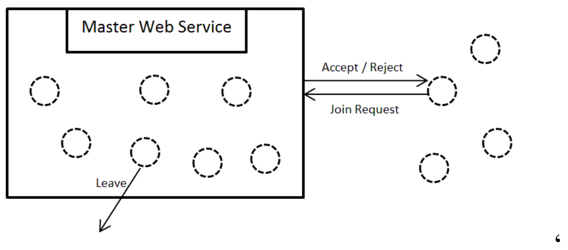


Figure 3.2: Web Services and A Grand Community.

web service. The goal is to have quality web services in the community so it stays stable and no other web services would have incentives to deviate and leave the coalition C. Therefore, a basic method would be to check the core of the coalition C considering all the current community members (all web services already residing within the community) and the new web service. This algorithm uses the *Shapley value* distribution method as described in Equation 2.1 to distribute the gain of $v(C)$ among all the members and then checks if the *Shapley value* payoff vector for this community having the characteristic function $v(C)$ is in the *core*. In the *Shapley value* payoff vector, the payoff for each web service ws_i is calculated based on its marginal contribution $v(C \cup i) - v(C)$ over all the possible different permutations in which the coalition can be formed, which makes the payoff distribution fair. Because of going through all the possible permutations of subsets of N, the nature of the *Shapley value* is combinatorial, which makes it impractical to use as the size of our coalitions grows. However, it is proven that in convex games, the *Shapley value* lies in the core [33, 62]. Thus, if the *Core* is non-empty, the payoff vector is a member of the *Core*. The following proposition is important to make our algorithm tractable.

Theorem 3.1. *A game with a characteristic function v is convex if and only if for all S, T, and i where $S \subseteq T \subseteq N \setminus \{i\}, \forall i \in N$,*

$$v(S \cup \{i\}) - v(S) \leq v(T \cup \{i\}) - v(T) \tag{3.3}$$

Proof. We first prove the "only if" direction:

1. "only if" direction:

Assume:

$$v(S \cup \{i\}) - v(S) \leq v(T \cup \{i\}) - v(T)$$

$$\rightarrow v(S \cup \{i\}) + v(T) \leq v(T \cup \{i\}) + v(S)$$

Considering $S \subseteq T$:

$$T \cup \{i\} = (S \cup \{i\}) \cup T$$

$$S = (S \cup \{i\}) \cap T$$

By setting $A = S \cup \{i\}$ and $B = T$ we have:

$$v(S \cup \{i\}) + v(T) \leq v(T \cup \{i\}) + v(S)$$

$$\rightarrow v(S \cup \{i\}) + v(T) \leq$$

$$v((S \cup \{i\}) \cup T) + v((S \cup \{i\}) \cap T)$$

$$\rightarrow v(A) + v(B) \leq v(A \cup B) + v(A \cap B)$$

Consequently, the game is convex.

2. "if" direction:

Assume the game is convex. Thus, for all $A, B \subset N$, we have:

$$v(A) - v(A \cap B) \leq v(A \cup B) - v(B)$$

By setting $S \cup \{i\} = A$ and $T = B$ where $S \subseteq T$:

$$v(S \cup \{i\}) - v((S \cup \{i\}) \cap T) \leq v(T \cup (S \cup \{i\})) - v(T)$$
$$\rightarrow v(S \cup \{i\}) - v(S) \leq v(T \cup \{i\}) - v(T)$$

□

Thus, in order to keep the characteristic function convex, new web services should have more marginal contribution as the coalition size grows.

Our algorithm works as follows. Given an established community with a master and some member web services, a web service would send a *join request* to join the community. Ideally, the *core* or *ε-core* stability of the group having this new member should be analyzed. As the normal core membership algorithm is computationally intractable, we exploit Proposition 3.1 and Equation 3.3 to check the convexity of our game having characteristic function where the new member is added. In the equation, let C be our community members before having the new web service join the community. Let i be the new web service, and then verify the equation for S, setting $S = T/W1$ where $W1$ is the set of all possible subsets of the set N having the size 1. We can relax the equation a bit by adding a constant ε to the left side of the equation. We call this method *Depth-1 Convex-Checker* algorithm. If the equation is satisfied for all $W1$, we let the new web service join our community, since the web service will contribute positively enough to make our new community stable. Since only subsets of size 1 are checked, the following Proposition holds.

Theorem 3.2. The run time complexity of Depth-1 Convex-Checker algorithm is $O(n)$.

By this result, we obtain a significant reduction from $O(2^n)$, which is the complexity of checking all possible subsets of N. In our second method, we use the same algorithm, but this time we set $W2$ to be the set of all possible subsets of size two and one of the

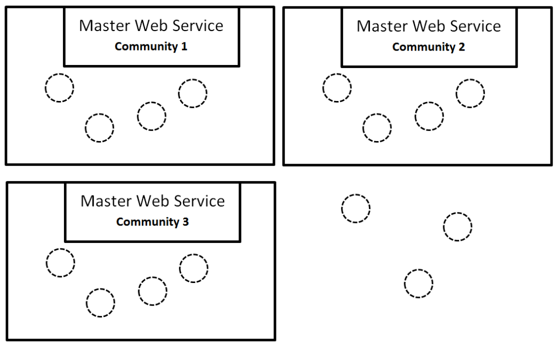


Figure 3.3: Web Services and Many Communities.

community C. We call this method *Depth-2 Convex-Checker* and its run time complexity is still linear:

Theorem 3.3. The run time complexity of Depth-2 Convex-Checker algorithm is $O(n^2)$.

It is possible to develop an algorithm that continues the verification of this condition against subsets of size 3, 4, etc. until the algorithm gets interrupted.

3.3.2 Web Services and Many Communities

In this scenario, we consider multiple communities managed by multiple master web services, each of which is providing independent request pools (see Figure 3.3). Identical to the first scenario, master web services form coalitions with web services. We use coalition structure formation methods to partition web services into non-empty disjoint coalition structures. As mentioned in Section 2.2.1, the used algorithms in [75, 33, 67] try to solve key fundamental problems of what coalitions to form, and how to divide the payoffs among the collaborators.

In coalition-formation games, formation of the coalitions is the most important aspect. The solutions focus on maximizing the social welfare. For any coalition structure π, let $v_{cs}(\pi)$ denote the total worth $\sum_{C\in\pi} v(C)$, which represents the *social welfare*. The solution concepts in this area deal with finding the maximum value for the social welfare over all the possible coalition structures π. There are *centralized* algorithms for this end, but these approaches are generally NP-complete. The reason is that the number of all possible partitions of the set N grows exponentially with the number of players in N, and the centralized algorithms need to iterate through all these partitions. In our model, we propose using a distributed algorithm where each community master and web service can be a decision maker and decide for its own good. The aim is to find less complex and distributed algorithms for forming web services coalitions [5, 29, 68]. The distributed merge-and-split algorithm in [5] suits our application very well. It keeps splitting and merging coalitions to partitions which are preferred by all the players inside those coalitions.

This merge-and-split algorithm is designed to be adaptable to different applications. One major ingredient to use such an algorithm is a preference relation or well-defined orders proper for comparing collections of different coalition partitions of the same set of players. Having two partition sets of players, namely $P = P_1, ..., P_k$ and $Q = Q_1, ...Q_l$, one example would be to use the social welfare comparison $\sum_{i=1}^{k} v(P_i) > \sum_{j=1}^{l} v(Q_j)$. For our scenario, we use *Pareto order* comparison, which is an individual-value order appropriate for our self-interest web services. In the Pareto order, an allocation or partition P is preferred over another Q if at least one player improves its payoff in the new allocation and all the other players still maintain their payoff ($p_i \geq q_i$ with at least one element $p_i > q_i$).

The valuation function $v(C)$ for this scenario is the same as *"Web Services and One Community"* scenario. However, in order to prevent master web services joining the same community, we set $v(C) = 0$ when C has either none, or more than one master web service as member.

In this scenario, as new web services are discovered and get ready to join communities, our algorithm keeps merging and splitting partitions based on the preference function.

The decision to merge or split is based on the fact that all players must benefit. The new web services will merge with communities if *all* the players are able to improve their individual payoff, and some web services may split from old communities, if splitting does not decrease the payoff of *any* web service of the community. According to [4], this sequence of merging and splitting will converge to a final partition, where web services cannot improve their payoff. More details of this algorithm and analysis of generic solutions on coalition formation games are described in [5].

3.3.3 Taxation, Subsidizing and Community Stability

We discussed *core* as one of the prominent solution concepts in cooperative games. Working together, completing tasks and generating revenue, agents need to distribute the gain in a way no agents would gain more by forming their own group. However, in most cases, the core of a game is empty, so we introduced the ε-*core* concept, where agents would only earn a minimal amount of ε by deviating from the coalition. Stability is an attractive property for communities. In addition, communities would benefit by having slightly more web services than the exact number of web services needed to satisfy the task rate cap. This is because there is always a possibility that the web services may leave the community or they may under perform and degrade the quality values they were initially performing with.

The solution we propose for communities to ensure stability is applying a tax ε, which is an amount of cost for those web services that decide to change communities (let us say from C to C'), which would make deviation a costly act. However, this would require all the community coordinators to agree on a same amount of taxation, being governed by some external entities; otherwise, web services would join communities charging the lowest amount of tax. Before deciding to change the community, each web service i has to be sure that the gain $g_i(C \to C')$ calculated in Equation 3.4 based on the Shapley values of i in the previous and new communities and the tax ε is positive, which means, what the web service would gain in C' is greater than what it gains in C and the tax it would pay if moving all together:

$$g_i(C \to C') = \phi_i(C', v) - \phi_i(C, v) - \varepsilon \tag{3.4}$$

Another viable solution we introduce to our scenario is to stabilize the game using external subsidies. The reason a game is not stable is that the community is not making enough revenue to allocate enough gain to the players. A community coordinator can subsidize its community with a constant coefficient value of λ. Obviously, with a big value for λ, it is always possible to stabilize the community. However, this can be a costly act for the community coordinators, so they are interested in the minimum subsidize value of λ making the community stable. This can be achieved by solving the following linear program:

$$\begin{aligned} &\min\ \lambda \\ &\quad \text{s.t. } \lambda v(C) > v(C') \ \text{ for all } C' \subset C \end{aligned}$$

Subsidizing or taxing in order to reduce the bargaining power of sub coalitions are called *taxation* [92] methods. We evaluate the effectiveness of these two methods experimentally through extensive implementations in the next section.

3.4 Experimental Results and Analysis

In this section, we discuss the experiments we performed for our scenarios to validate the applicability and performance of our proposed methods in realistic environments. An XML

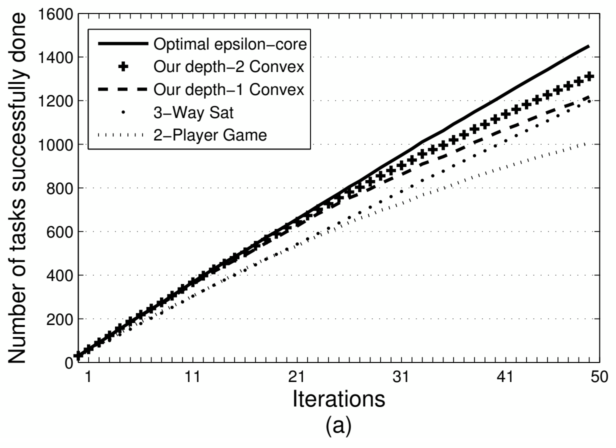
SOAP based messaging system was implemented. We created a pool of web services and populated most of their *QoS* parameters from a real world web service dataset [2][1]. To test our methods, we formed around 10,000 random coalitions consisting of 3 to 160 web services. In average, the communities were populated by 60 web services. We implemented the scenarios using Java and executed the experiments on an Intel Xeon X3450 machine with 6GBs of memory.

One of the key criteria reflecting the performance of web service coalitions is the user satisfaction. User satisfaction can be measured in terms of quality and quantity of requests (or tasks) successfully answered by the communities. We initiated the communities with few web services, then let rejecting and accepting random web services go for a short number of iterations. After that, we started the request distribution for the communities and let them allocate requests among member web services. Thereafter, we measured the average output performance of tasks in communities following different methods.

Figure 3.4 depicts the results of optimal *ε-core*, *Depth-1 Convex-Checker*, *Depth-2 Convex-Checker*, *3-Way Satisfaction* [46], and *2 Player Non-Cooperative* [41] methods in *one grand community with many web services* scenario.

For the *optimal core* method we have used the well known *ε-core* method as the taxtation method to relax the core condition to help communities, attract web services. We have assigned ε to 15% of total community worth, $\varepsilon = 0.15 \times v(C)$, which allows subsets of the coalition to gain maximum 15% of $v(C)$. In the *optimal ε-core* method, we capped the coalition size to 25 web services, since the method is computationally intractable as number of web services increase and anything more than that would make it impractical to run in our simulations. In the other methods, there were no cap on size of the community and we had communities of size 60 web services at some points. In this scenario our

[1]The implemented environment includes the QWS dataset by Eyhab Al-Masri and Qusay H. Mahmoud freely available at: `http://www.uoguelph.ca/~qmahmoud/qws/`.



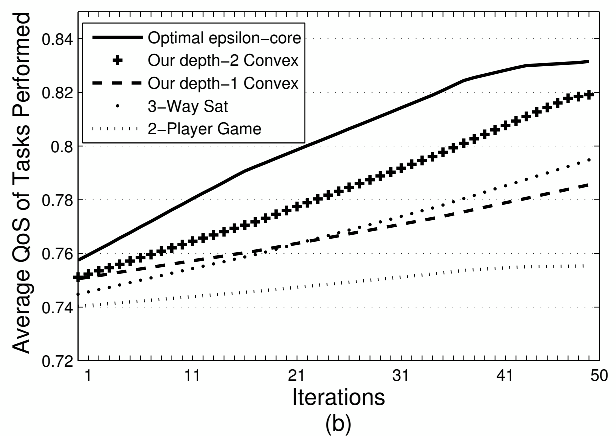


Figure 3.4: Part (a): Cumulative number of requests successfully done. Part (b): Average QoS of requests performed.

community receives 30 tasks on average per iteration, from users. The community, after the task distribution process on each iteration, will reevaluate QoS metrics of its members and can check for new membership requests. Web services may join or leave the community between iterations. The results show that our *depth-2 convex checker* method is performing better compared to the other methods and its performance is close to optimal *ε-core* method. Our *depth-1 convex checker* and the *3-Way Satisfaction* method, are also performing well.

As mentioned in Section 3.1, the concept of *core*, assumes no coalition of players can gain anything by deviating, which is a fairly strong requirement, and that is why the notion of *ε-core* was introduced. Least-Core $e(G)$ of a game G, is the minimum amount of ε so

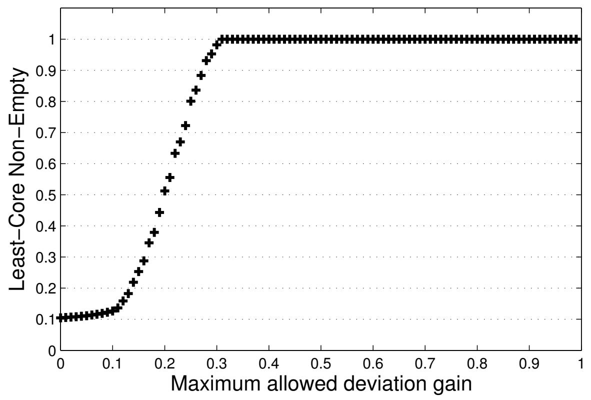


Figure 3.5: Analysis of ε*-core* set non-emptiness, for different values of ε.

that the core is not empty. We evaluated the non-emptiness of ε*-core* set using the valuation function and a set of web services. We picked random number of web services from the dataset and formed around 10,000 random coalitions consisting of 3 to 26 web services. We choose 26 as the maximum number of members in our coalition since it is computationally very complex for larger coalitions to verify whether ε*-core* set is empty or not. Also instead of considering ε amount of constant deviation in ε*-core* definition (Equation 2.4), we similarly defined *relative* ε*-core* concept where no coalition would benefit more than $\varepsilon \times v(C)$ by deviating. We set ε between 0 and 1 and verify the *relative* ε*-core* set non-emptiness. The results in Figure 3.5 illustrates that almost 10% of our random web service coalitions have non-empty *core* solution and ε*-core* solution is *always* non-empty when we let agents gain only 30% more of $v(C)$ by deviating.

One of the properties of coalition structure formation algorithms in our second scenario is that they partition web services with low throughput rate so that they usually join coalitions with less request rate. Since the characteristic function $v(C)$ and the fair Shapley payoff vector is proportional to web services' contribution, the web services with small contribution will get paid much less in communities having web services with high throughput. On the other hand, according to the valuation function $v(C)$, web services with high

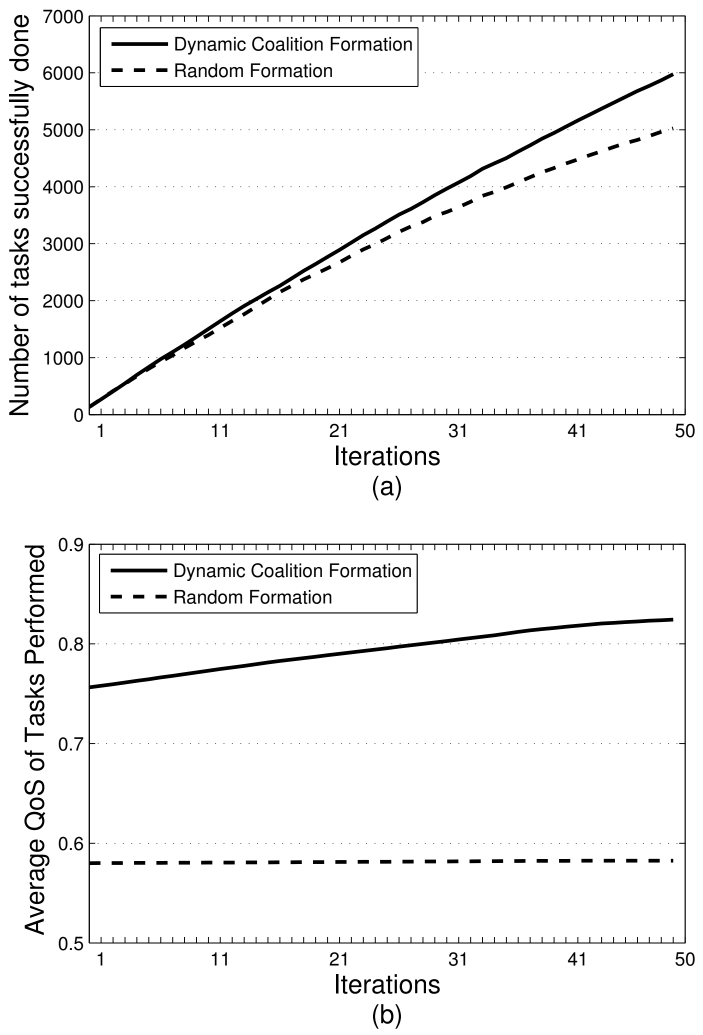


Figure 3.6: Part (a): Cumulative number of tasks succesfully done. Part (b): Average QoS of tasks performed.

throughput will not contribute well to communities with low amount of user requests (low market share). The strong web services are likely to deviate from weak coalitions, joining a stronger one, which makes the initial coalition unstable.

In Figure 3.9, we compare our *Web Services and many Communities* scenario with a method which ignores QoS parameters and forms coalitions by allowing web services to join only if they have enough requests for themselves. In other words, web services can join a community when the request rate is less than the throughput of all the member

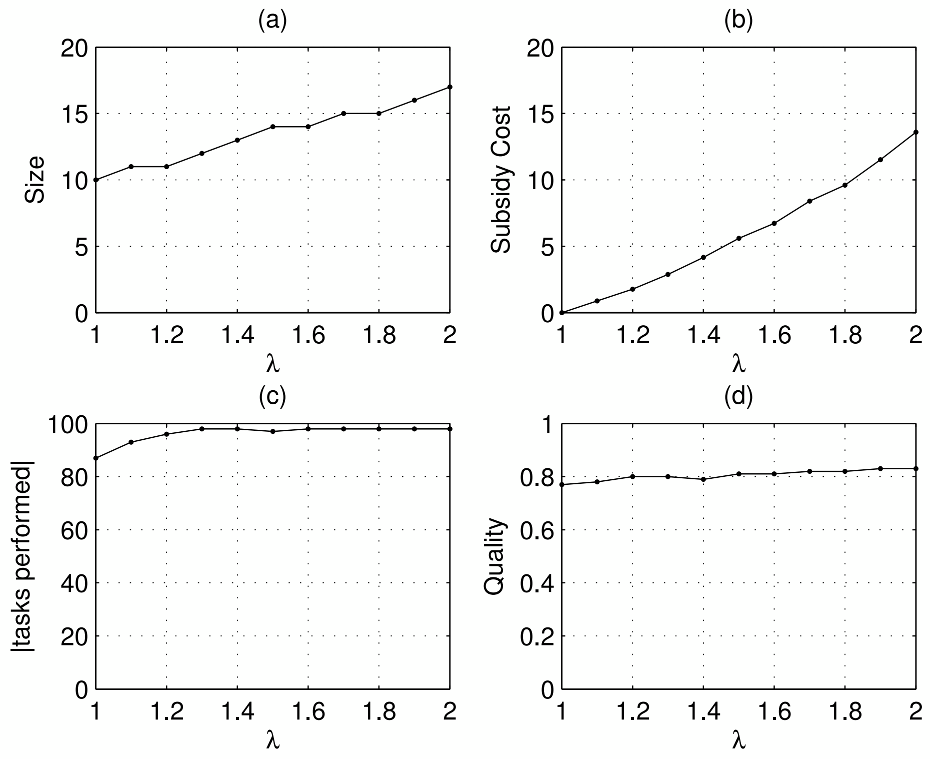


Figure 3.7: Analysis of community subsidizing coefficient λ on average community size (a), cost (b), number of tasks performed (c), and average quality of service of tasks performed (d).

web services. We name this method *Random Formation* and use it as a benchmark for our QoS-aware coalition formation process. In this scenario, each user individually generates randomly between 0 to 10 number of tasks per iteration, then the users target a community and direct their requests to the chosen community. As the results illustrate, our method forms better coalitions of web services improving performance and satisfaction for both web services and coalitions.

As mentioned in Section 3.3.3, a solution to help the community stabilize is to subsidize the community by a relative coefficient (λ) so the value of $\lambda v(C)$ is divided among the community members. We have analyzed the effect of subsidizing and the cost it incurs to our web services communities. Figure 3.7 shows the results. In this experiment, we have set a community with input task rate R_C of 100 and having web services throughput rate Th_{ws} values from a normal distribution with average 10 tasks per iteration and standard deviation 2. Part (a) shows the community size increases in a linear fashion as (λ) increases. However, the cost (Part (b)) is having a slight exponential growth rate since, not only (λ) increases, but also the size of the community is increasing slowly. Therefore, subsidizing can be costly for larger number of λ values. Part (c) depicts the number of tasks done by the community per iteration. It is obvious that with λ value of 1.3, which is 30% of the community valuation, the number of tasks done almost reaches the input task rate cap of 100 tasks per iteration. The average quality of tasks also has a slight increase since the community will be able to afford better and more web services to join the community (Part (d)). These results show the effectiveness of our subsidizing method and its impact on the QoS. In fact, using more than 30% of the community valuation as subsidy is not very effective and is costly to perform.

In our next scenario, we have introduced the new instability variable τ ranging from 0 to 1, 0 meaning web services having no instability issues and will perform as they claimed until the end of the experiment, and 1 meaning very unstable web services, which will stop functioning on the first iteration of the community distributing tasks. Figure 3.8 illustrates the results of our experiment having web services with average instability values of 0 to 0.5 and having relative subsidy value λ of 1, 1.3, 1.6, and 2. The *Cost/Income* charts on the right column show that having subsidy value of 1.3 incurs the least cost and increases the community income significantly. Subsidy values of 1.6 and 2 yield high cost to the community and only slightly increase the community revenue. Moreover, the role of subsidizing is much more obvious when we have unstable web services. In scenarios where web services are 100% stable, the subsidizing cost will hardly be compensated by the community revenue.

web services are stable, will not leave the group, and will fulfill their promised QoS for a good period of time. However, in real world scenarios of web services, this is not always the case. This is the reason why the community coordinator would be interested in paying web services in order to keep the group reliable from the end user's point of view.

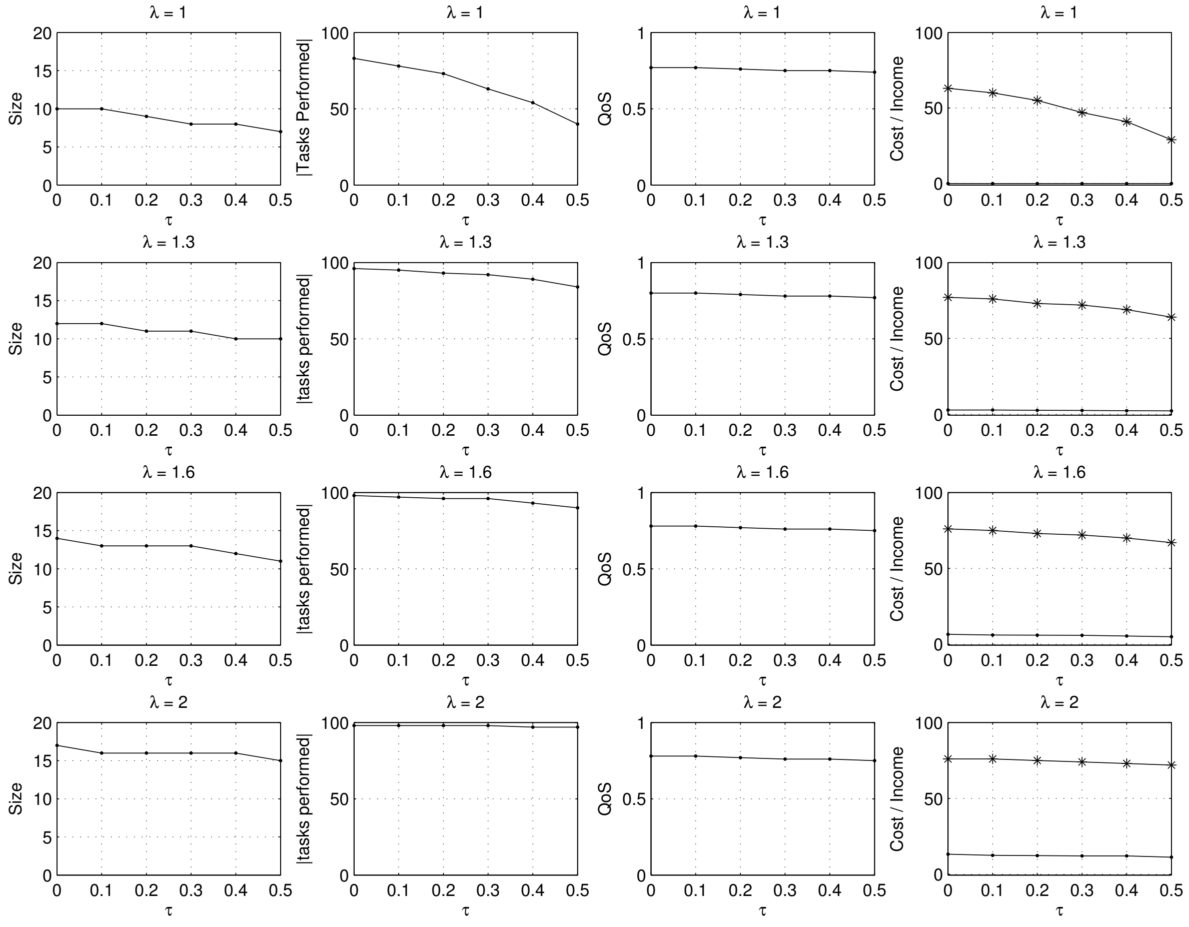


Figure 3.8: Analysis of community subsidizing coefficient λ having web service different stability levels of τ on average community size, number of tasks performed, average quality of service, and average cost/income of communities.

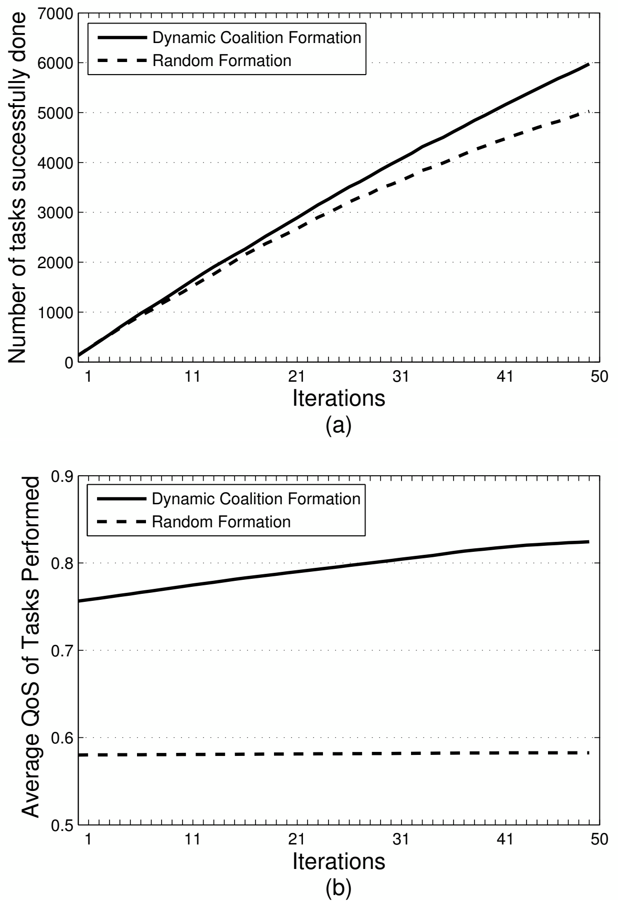
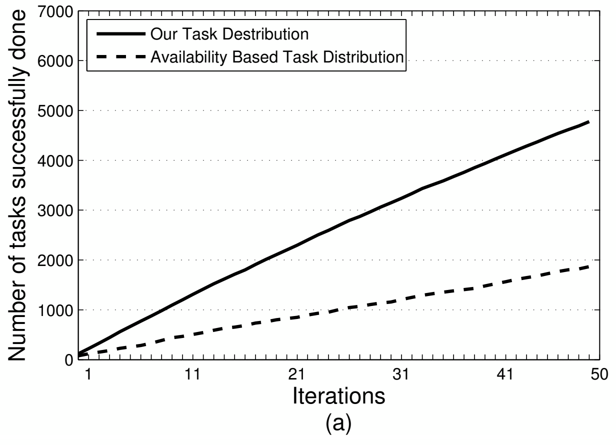
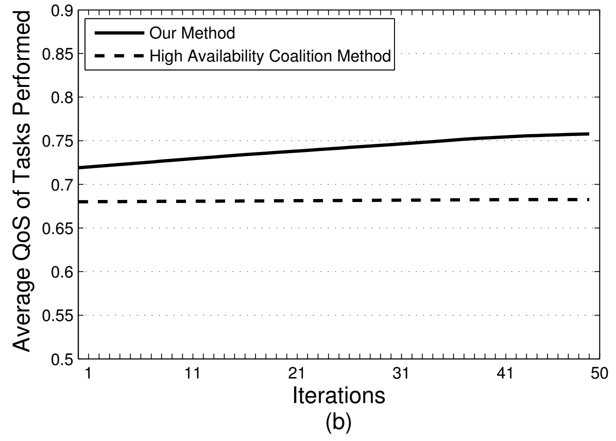


Figure 3.9: Part (a): Cumulative number of tasks successfully done. Part (b): Average QoS of tasks performed.

In Figure 3.9, we consider *Web Services and many Communities* scenario and we compare our dynamic coalition formation solution with a method which ignores QoS parameters and forms communities by allowing web services to join only if they have enough requests for themselves. In other words, web services can join a community when the request rate is less than the throughput of all the member web services. We name this method *Random Formation* and use it as a benchmark for our QoS-aware community formation process. In this scenario, each user individually generates randomly between 0 to 10 number of tasks per iteration, then the users target a community and direct their requests to the chosen community. As the results illustrate, our method forms better communities of web services improving performance and satisfaction for both web services and communities.

Finally, in our last experiment, we compare our model with the solution proposed in [48], which we call *High Availability Coalition* model. In this method, the community valuation function focuses on the community availability as main consideration. The community formation model used in this method is very different from ours, but we have been very careful to make the experiment environment as fair and similar to ours as possible. We limited our maximum community size to 5 in order to have communities with almost the same size as in [48]. In the High Availability Coalition model, the authors have used web services as backups rather than active collaborative players, and those web services only get a task when the first web service in an ordered chain fails to perform that task. Part(a) of Figure 3.10 shows that with our method, the number of tasks successfully done is higher with a rate of three times more than the High Availability Coalition model thanks to the cooperative behavior of web services and the task distribution process of our algorithm. This result shows that using web services as backups, and not as real collaborative players results in a considerable waste of web services capability since services have very low chance of getting jobs and its the primary web service (the first in the coordination





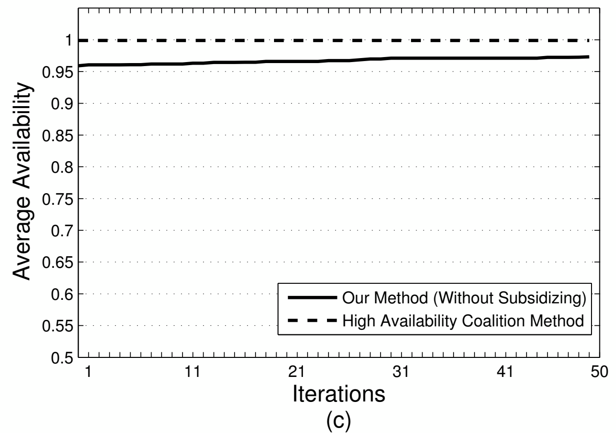


Figure 3.10: A comparison between our community model and the High Availability Coalition model from [48]. Part (a): Cumulative number of tasks successfully done. Part (b): Average QoS of tasks performed. Part (c): Average community service availability.

chain) which does most of the work. As shown in Part (b), the average quality of service of tasks performed using our solution is also higher since our method considers all quality of service metrics used. Part(c) shows the availability of communities from the end user's point of view. The High Availability Coalition model has almost 100% uptime since web services are used as backups, so the chance of job failing is getting reduced significantly as community members increase. In our method, we have more chance of failure for each web service. However, with some subsidies and by hiring a few more web services, the chance of failure of web services in our communities can be lowered.

3.5 Summary

In this chapter, we proposed a cooperative game theory-based model for the aggregation of web services within communities. The goal of our services is to maximize efficiency by collaborating and forming stable coalitions. Our method considers stability and fairness for all web services within a community and offers an applicable mechanism for membership requests and selection of web services. The ultimate goal is to increase revenue by improving user satisfaction, which comes from the ability to perform more tasks with high quality. Simulation results show that our, polynomial in complexity, approximation algorithms provide web services and community owners with practical and near-optimal decision making mechanisms.

In this chapter, we addressed the research questions *R1, R2* and *R3* that were introduced in section 1.3. We assumed our web services and communities have complete knowledge of all the other web services and their parameters. In the next chapter, we propose distributed decision making model which can perform in scenarios where information is incomplete.

Chapter 4

Distributed Decision Making for Dynamic Formation of Web Services Communities

4.1 Introduction

Given the dynamic and unpredictable nature of the Internet, delivering high quality services is a critical and challenging issue. One practical solution towards delivering such quality services is utilizing intelligent decision making agents. These agents aim at maximizing their gain by exploring the best ways to provide services that satisfy end users [89, 52, 26, 91, 36]. However, agent-based web services are functionally limited in the sense that they cannot handle a large number of requests at the same time without compromising the quality of service provided. Recent developments have attempted to shift web services from simple models, consisting of individual components, to models made up of autonomous and group-based components that share common goals. In group-based models, interaction, composition, and cooperation are the key challenges that directly impact the group's overall performance in achieving common goals [69, 16, 12, 20]. To that end, we see the emergence of web service *communities*, which consist of grouping services with similar functionalities but distinct nonfunctional properties [89, 52, 66, 59]. A community of web services runs continuous performance assessment functions that regulate web services' interactions and manage their composition and cooperation.

Web Services communities have the advantages of facilitating web service discovery and providing better quality of service compared to individual services. Communities act as abstract web services, communicating with external entities via the same standard protocols that a normal web service employs. The difference is that communities regulate the service process via sophisticated internal communication protocols, thereby providing services based on the combined efforts of a number of web services. The downside to communities is the complexity of management involved in finding and inviting adequate individual services and managing the overall quality of the combined work of several services. When interacting with a community of web services, users send their requests to the coordinator of the community, which plays the role of community representative or access point. The community coordinator is responsible of receiving tasks and delivering services. Moreover, as community representative, it verifies the credentials of new web services before accepting them into the community and kicks services that could harm the value of the community.

Challenges and Problem Statement. In recent work, communities of web services have been proposed in order to facilitate discovery of web services, improve the Quality of Service (QoS), and help individual services find better market share and opportunities [89, 52, 66, 59]. However, two important challenges are to be addressed: 1) choice of the best web services during community development from the community perspective; and 2) choice of the best community to join from the web service perspective. The advocated solutions [52, 51, 84, 48, 47, 41, 46] have attempted to address these challenges. However, those solutions have two main limits:

1. The solutions consider the architecture of centralized management for communities where most of the decisions are made by the centralized coordinator. The problem is that in real world scenarios, decisions made by independent service providers are highly distributed.

2. The solutions either propose complex algorithms [48, 46, 8] to find the optimal strategy to follow, or oversimplify the problem by eliminating important parameters and using approximation techniques to make the algorithms tractable [48].

These approximation methods sometimes negatively influence the outcome because simplifying the constraints may cause important aspects of the problem to be ignored. For instance, instead of calculating the gain distribution using the adequate, but complex shapley-value method, the authors is [48] propose a simple egalitarian way of distributing gain, which completely ignores the gain generated from collaborative work of sub-communities. Other categories of related work, for instance [48, 41, 53], restrict the decision process within the community coordinator, so other members of the community are not effectively involved. In [8], we proposed a cooperative game-theory-based model for aggregating web services in communities. A centralized decision maker in communities, based on a complete knowledge of available web service quality metrics and performance, has been used to form optimal and stable communities that maximize individual and group income. However, centrality and complete information are strong assumptions, which are not very compatible with real business scenarios.

Contributions. In this chapter, we introduce DDM, a Distributed Decision Making model for community formation that regulates web service agents' decision making process in terms of cooperating and deciding which group to join and which service to invite for joining. Unlike existing work on community formation, our decision model is extracted from a data model in the form of information obtained from a large number of web services regarding their single and cooperative utilities as well as environmental parameters such as demand, service quality, etc. The generated decision tree improves agents' understanding of the environment and how to select actions that lead towards maximizing their utilities. The advantage of this approach is that the tree, which is initially created from the past data, reflects a comprehensive vision about agents' attitudes in terms of their action selection based on their past experiences. Moreover, the tree is getting continuously updated based on both new received feedback and the outcome of chosen actions. This continuous update makes the approach adapted to any change in the environment. The decision model provides web services with enough information which helps those services efficiently decide and predict the outcome of their different possible collaborations. This model works in a distributed manner in which services are self-sufficient in their decision making and do not rely on a centralized decision making process. Our findings show that communities of web services can efficiently find the appropriate web service to invite for cooperation as well as allowing a single web service to find the best communities to join. The proposed model can be seen as a recommneder system that suggests beneficial actions for both communities and single services. Communities can consider the decision model and analyze the characteristics of different individual web services and make prudent decisions when inviting a web service to join or accepting a join inquiry initiated from a web service. In general, DDM equips web services with efficient methods for foreseeing how their choices will impact both their short-term and long-term goals; therefore, opting for the best decision available.

To effectively generate the decision model for web services, we used a real dataset to extract web services' individual characteristics and used them to measure outcomes when these services cooperate with one another. The dataset has been extracted from real-world

QoS evaluation results from 142 users on 4,532 Web services during 64 different time slots. Combining the available data based on each web service point of view on different time slots, we acquired 5 different unique features for those 4,532 web services. By engineering and extracting these features, we gathered functional and cooperative features for both individual web services and communities in different time slots. We were able to investigate the path a web service might take to achieve the best utility out of effective interactions with others. All the paths and outcomes are labeled to be utilized in the training model. Using cross validation sets, web services are able to compute the optimal hypothesis function (using logistic regression) that can be used to predict outcomes of cooperative work with other individual web services or communities. Our findings show that web services equipped with DDM have by far better outcomes than the ones that either do not cooperate or randomly find communities to join.

4.2 Challenging Issues

In this section, we introduce the challenges behind community formation.

4.2.1 The Join Challenge

It has been showed in [52, 48, 8] that web services can increase their overall utility by collaborating with other web services within communities. This collaboration provides them with better ways of sharing resources and having higher reputation, greater market share and wider visibility. Web services and communities come with different quality metrics, and the long-term outcome depends on these metrics.

The goal of all parties involved in the community is to maximize their long-term outcome while they are operating as part of the community. Web services need to be equipped with a selection strategy to choose from the different possible collaboration groups they can form as well as an estimation method for evaluating the long-term gain of joining different possible communities. Web services need to experiment with different possible collaborative groups in order to estimate their gain over time. However, with a high number of possible communities, it is not possible to test collaboration with random web services. Even if a linear approximate function for estimating utility based on community web services' parameters is adopted, the exponential [1] growth rate of the possible number of partitions of web services into communities would make any brute-force type algorithm for the best community selection strategy intractable and impractical in real-world application settings.

4.2.2 Join Consequences

It is worth mentioning that a *join* event takes place as a result of interaction between two parties that are looking to expand their collaborations. All actions are chosen in an attempt to enhance the overall outcome. However, the selected action may result in decreasing the overall utility in the long run. This is the case when a single web service joins a community, but the complex process of task allocation eliminates the visibility of that service, which stays idle within the community. This makes the join action of that service a bad decision. The same event might be beneficial for the community, as it hosts a new web service that can engage in performing a new coming task. But overall, in this particular case, if the new web service stays idle for a long period of time, neither side will benefit from collaborating with the other and the join event will result in negative consequences for at least one side's utility.

The more common scenario is when both parties benefit from the joining of a web service to a community. This joining action is then rational as both the web service and

[1]Bell number: `http://en.wikipedia.org/wiki/Bell_number`

community enhance their utilities. However, the community may not be the best choice for the web service. In other words, the web service could have joined a better community if it had enough and accurate knowledge about the surrounding environment. Since the community does enhance its utility, the web service could stay with that community, which results in a non-optimal increase in web service's utility. In the following section, the proposed model provides solutions that effectively address the aforementioned challenges.

4.3 Model Components

In this section, we discuss the parameters that we use in the rest of the chapter. Then, we present the task distribution and revenue model of our distributed web services communities.

4.3.1 Internal Features

With a group of web services having identical or similar functionalities, QoS metrics provide nonfunctional characteristics for optimal candidate selection. Web services quality metrics have been studied and analyzed in various proposals, for instance in in [6, 61, 90]. In this chapter, we adopt the most representative QoS properties of those services that highly influence their utility.

Let $C = \{ws_1, ws_2, ..., ws_n\}$ be a community with n web services. We define the following features for the group of web services based on their functional parameters:

- *Throughput* is the rate at which a service can process requests. QoS measures can include the maximum throughput or a function that describes how throughput varies with load intensity. Throughput is a positive real number. For a given community C, the expected throughput value (Th_C) can be estimated as the summation of throughput of all the service members Th_w $(w \in C)$:

$$Th_C = \sum_{w \in C} (Th_w) \tag{4.1}$$

- *Availability* is the percentage of time that a service is operating. It is computed as the probability that the service operation is accessible. Availability of a web service A_w is a real number in the range $[0,1]$. For a community C, the expected availability (A_C) considering the members operate in parallel (independently from each other) can be estimated as:

$$A_C = 1 - \prod_{w \in C} (1 - A_w) \tag{4.2}$$

- *Execution Time* is the time a service takes to respond to various types of requests. Execution time is usually measured in milliseconds and can be affected by load intensity, which can be measured in terms of arrival rates (such as requests per second) or number of concurrent requests. This internal feature is a positive integer. For a typical community C, the expected execution time Et_C can be estimated as the execution time of the bottleneck service which is the service with the slowest execution time Et_w:

$$Et_C = max_{w \in C}(Et_w) \tag{4.3}$$

We normalize the range of these features so that each feature contributes proportionally to the final utility outcome value. We adopt the *standardization* method consisting of subtracting the *mean* from each feature, then dividing the subtraction result by the *standard* *deviation*.

4.3.2 External Features

The quantitative values of quality metrics need some benchmark values to represent their goodness. In fact, without some benchmark values, it would be difficult for web services to identify their performance quality at any specific value of these metrics. Therefore, we introduce two external features for assessing web services' estimate with regard to their standing among other web services.

- *External Parameter 1* ($Exp1_i$ where i is a community or a web service) is an estimate of how close the community's or the web service's *execution time* is to the best execution time in the whole system. It is the difference between a community's or a web service's *execution time* metric and the minimum value of execution time of all the other communities or web services. The smaller the value the better the external feature compared to other peers. In other words, small value of $Exp1_i$ means i is among the best communities or services in the system.

$$Exp1_i = Et_i - Et_{min} \tag{4.4}$$

- *External Parameter 2* ($Exp2_i$ where i is a community or a web service) is a comparison of the community's or the web service's rate of performing tasks to the best rate in the system. It is the difference between a community's or a web service's *throughput* metric and the maximum value of throughput in the system. As for $Exp1_i$, the smaller the value the better the external feature.

$$Exp2_i = Th_{max} - Th_i \tag{4.5}$$

4.3.3 Task Distribution

Communities of web services usually employ an implementation of Contract-Net protocol for task distribution, in which services bid on incoming tasks, and receive some of the tasks for which they bid [53, 30]. In our model, our community members would try to distribute tasks based on their capabilities and the QoS parameters provided by the web services. We use a slightly modified *weighted fair queuing* method to distribute tasks among community members. The goal is to allocate incoming tasks to web services with a rate matching the throughput value of Th_w for each web service w. In the *weighted fair queuing* method, the input flow is multiplexed along different paths. However, in our model, if the rate of incoming tasks is less than the community's total throughput (Th_C), which is the summation of throughput values of the web services in the community, some of the input tasks will be queued and served with a delay. When the incoming task rate is less than the throughput of the community, the *weighted fair queuing* algorithm assigns a weighted task rate of $Itr \times \frac{Th_w}{\sum_w Th_w}$ for each web service w within the community, where Itr is the input task rate.

While distributing tasks, the community can verify the performance, throughput and quality of service of tasks being performed by web services. The community can assess if those web services are capable of performing the number of tasks they advertised. If for any reason, there is a decline in the quality metric or throughput, the community can consider the new values as a benchmark for future performance calculations, and penalize the suspicious web services. This way, players will have incentive to truthfully disclose their actual capabilities in order to maximize profit from the community and to avoid being penalized. In addition, the system should be dynamic enough to detect and react to web services' quality metrics variation, as over time, web service metrics may degrade or improve, changes to which the community should adjust.

4.3.4 Community Revenue

Communities and web services earn revenue by performing tasks. The total gain is a function of the quality and rate of performing tasks. The utility of a collaborative group of services U_C (i.e., the revenue of the community) is a function of internal and external parameters:

$$U_C = f(A_C, Et_C, Exp1_C, Exp2_C, Th_C) \tag{4.6}$$

where f is increasing in A_C and Th_C and decreasing in $Et_C, Exp1_C$ and $Exp2_C$. An example of this function is given in Equation 4.7:

$$U_C = \big((\alpha \times (A_C - Et_C) - \beta \times (exp1_C + exp2_C)\big) \times Th_C \tag{4.7}$$

The α and β parameters are internal and external weight coefficients. Small values for execution time and external parameters ensure better performance, which justifies their negative coefficients. The result is then multiplied by the throughput value Th_C, since communities are performing tasks with Th_C rate.

Theorem 4.1. *The function given in Equation 4.7 satisfies the properties of f.*

The proof of this theorem is straightforward by simply calculating the partial derivative ∂f with respect to the different variables.

The estimation of the utility can be improved, especially in cases where the input task rate is high and services are experiencing high task loads. The *weighted fair queuing* method of task distribution would distribute tasks based on the individual throughput (Th_w) value of services within community. In fact, services having higher throughput affect strongly the overall utility of the community because they would take on proportionality more tasks. The improved utility is given as a function of individual internal and external parameters:

$$U_C = g_{w \in C}(A_w, Et_w, Exp1_w, Exp2_w, Th_w) \tag{4.8}$$

where $g_{w \in C}$ is increasing in A_w and Th_w and decreasing in $Et_w, Exp1_w$ and $Exp2_w$. An example of this function is given in Equation 4.9:

$$\begin{aligned} U_C = \sum_{w \in C} \Big(\big(\alpha \times (A_w - Et_w) \\ - \beta \times (Exp1_w + Exp2_w) \big) \times Th_w \Big) \end{aligned} \tag{4.9}$$

The following theorem holds:

Theorem 4.2. *The function given in Equation 4.9 satisfies the properties of* $g_{w \in C}$.

4.4 Decision Making Mechanism

In this section, we describe our data extraction process and the methodology used to equip web services and communities with a decision making mechanism. In this methodology, we first present the data extraction and engineering process and then we evaluate the decision making mechanism for web services in community settings. Figure 4.1 summarizes the steps performed in DDM from the input data to the generation of decision making profiles for web services and communities. The objective is to use the input data to build a decision tree for each service and community included in the data set, which will be be served as a benchmark for other services and communities in their decision making mechanism. The decision tree is made up by training the real data obtained from operating web services and extracting features related to their performance, either alone or as part of a joint effort

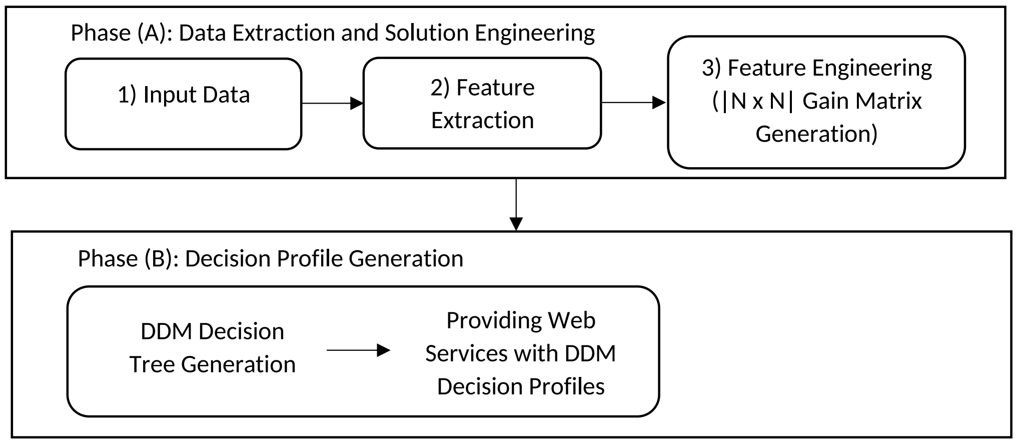

Phase (A): Data Extraction and Solution Engineering
1) Input Data
2) Feature Extraction
3) Feature Engineering (|N x N| Gain Matrix Generation)
Phase (B): Decision Profile Generation
DDM Decision Tree Generation
Providing Web Services with DDM Decision Profiles

Figure 4.1: A summary of DDM decision profile generation steps.

with other web services. The ultimate objective is to propose for each web service and community the best joint decision about forming a group that maximizes every one's utility. The DDM's steps are explained in the following sections.

4.4.1 Data Extraction and Solution Engineering

A. Input Web Services Data

Each web service is associated with a number of quality metrics that reflect its non functional parameters. These web services operate in an online environment and are continuously assigned tasks to handle. We used the web services data set provided in [90]. The raw data provides real-world QoS evaluation results from several users on 5,825 web services over 64 different time frames[2]. Using this data, we built a synthetic data set that contains features of a large number of web services and communities in different time intervals. The goal is to use the data set to train a decision-making model that adopts the trend of joining a community and use the model to predict/find the appropriate community for other web services.

[2] `http://www.wsdream.net/`

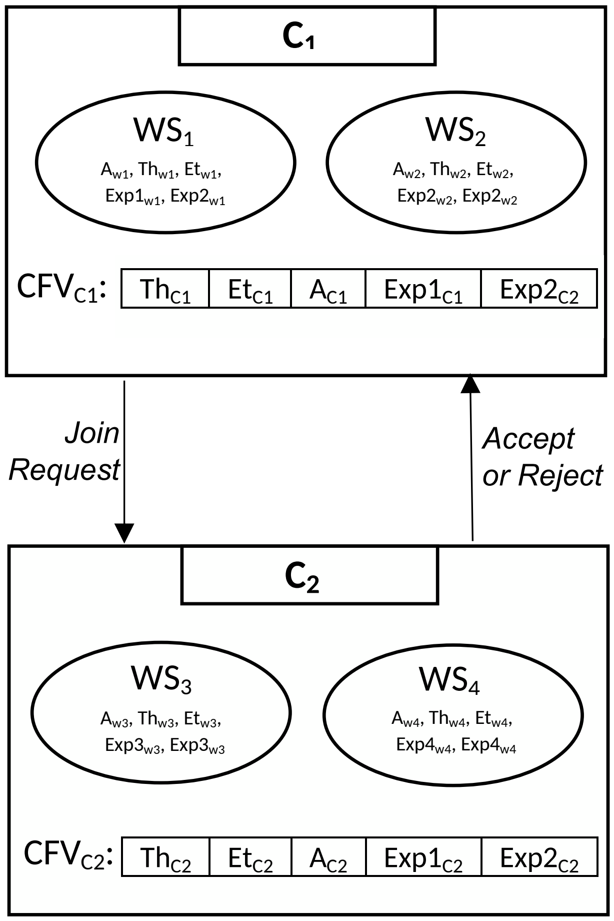


Figure 4.2: Communities with different properties of web services actively looking for other communities to collaborate with.

B. Feature Extraction

By processing the data provided for each web service over different time slots, we obtain the three internal quality features introduced in Section 4.3.1: *throughput*, *availability* and *execution time* and the two external features discussed in Section 4.3.2. In fact, web services and communities are represented using feature vectors of these five internal and external features.

We formulate a *Community Feature Vector (CFV)* as $CFV_{<C>} = [f_1, ...f_5]$ having a community of k web services ($C = \{ws_1, ...ws_k\}$, $k \geq 1$)[3]. The features f_1 through f_5

[3]A web service is considered as a community of one web service.

represent the *execution time*, *throughput*, *availability* and the *external parameters 1 and 2* respectively. A set of communities, with their feature vectors and utilities evaluated, provides our algorithm with a raw training data set. We call this set of communities the *template vector CS*, and the set of feature vectors associated with the *template vector* is referred to as the *community feature vector set (CFVS)*. Figure 4.2 depicts web services and communities looking to form new groups in order to improve their utility gain.

C. Feature Engineering

Let $CFVS = \{CFV_{<C_1>}, \ldots, CFV_{<C_N>}\}$ be the community feature vector set with N communities. Based on the $CFVS$ set, we create an $|N \times N|$ gain matrix $gain^t$ for each time slot t. Each entry $gain^t_{n,m}$ corresponds to a utility gain of community C_n when it joins community C_m. This gain is computed as follows: $gain^t_{n,m} = U^t_{C_n \cup C_m} - U^t_{C_n}$ where $U^t_{C_n \cup C_m}$ and $U^t_{C_n}$ are the utilities at time t computed using Equation 4.9. Evaluating the utility gain for all entries of the $gain^t$ matrix is a computationally heavy process when N, the size of the feature vector set, is large. Therefore, this size should be chosen carefully.

Table 4.1: An example of *gain* matrix for 3 different communities and their combinations.

	<348>	<1934>	<2117>	<348, 1934>	<1934, 2117>	<348, 1934, 2117>
<348>	-	0.282708	1.027081	0.282708	18.027081	18.027081
<1934>	-2.637483	-	6.969072	-2.637483	5.509583	4.387725
<2117>	5.027081	2.969072	-	5.509583	2.969072	5.509583
<348, 1934>	0.0	0.0	-3.851432	-	-3.851432	-3.851432
<1934, 2117>	2.969072	0.0	0.0	2.969072	-	2.969072
<348, 1934, 2117>	0.0	0.0	0.0	0.0	0.0	-

Each community is provided with the corresponding row of data from the $gain^t$ matrix. Basically, C_i is provided with the data in row i of this matrix, which reports all the possible utility values C_i can gain by joining different communities. By ordering the utility gain values of the row, each community is equipped with an ordered set of preferences over other communities it can join. We define $\geq^t_i$ as the preference order of community i at time

t.

Let $C_1 \geq_i^t C_2 \geq_i^t \ \dots \ C_{i-1} \geq_i^t C_{i+1} \geq_i^t \ \dots \ C_n$ be an ordered sequence of preferences for community C_i at time t. Based on this sequence, we define $K^t(C_i,k)$ as a set of the k most preferred communities of community i at time t.

$$\begin{aligned} K^t(C_i,0) =& \emptyset \\ K^t(C_i,k) =& \Big\{ C_x | C_x \geq_i^t C_y \ \forall C_y \in CS \ \wedge \ C_x \neq C_y \ \wedge \ C_y \notin K^t(C_i,k-1) \Big\} \end{aligned} \tag{4.10}$$

Based on $K^t(C_i,k)$, we define a set of communities C_j for C_i which are the k most preferred communities for C_i and C_i belongs to the k most preferred communities of C_j. This basically yields the preference in both sides.

$$L^t(C_i,k) = \Big\{ C_j | C_j \in K^t(C_i,k) \ \wedge \ C_i \in K^t(C_j,k) \Big\} \tag{4.11}$$

Table 4.1 illustrates an example of a *gain* matrix for 3 different communities and their combinations. Each row shows the gain the community can achieve by collaborating with other 5 communities. In this example, for community $< 348 >$ we have:
$K(< 348 >, 1) = \{< 1934, 2117 >\}$ and
$K(< 348 >, 2) = \{< 1934, 2117 >, < 1934 >\}$
Since $< 348 >$ is the best preferred community of $< 1934, 2117 >$ and vice versa, therefore $L(< 348 >, 1)$ is not empty and contains the community $< 1934, 2117 >$.

Using the *gain* matrix and the mentioned preference ordering relations, we are able to build a decision tree where the list of possible communities to join and their expected utilities are set. In addition to the best choice, web services have access to other ordered choices and can look for the second best or third best if their first try is rejected by the target community. This aspect is analyzed in more detail in the following section, in which we launch experiments and investigate the effectiveness of the use of a decision tree with different decision layers in joining other communities and enhancing the overall utility.

4.4.2 Decision Profile Generation

Our goal is to create a decision making profile for each community in the $CFVS$ set. We are creating an environment where the communities can experience the outcomes of different strategies. The result will be a decision tree of the feasible and utility-increasing moves over time. The root of the decision tree represents a community in the $CFVS$ set, and the other nodes represent the communities resulting from the parent node's action of joining them along with their feature values and expected utility.

We let communities pick the best communities maximizing their utilities over different time frames. At time $t = 1$, we let each community in the $CFVS$ set choose the best community, which is a single community in the set $\{C_j\} = K^t(C_i, 1)$. If community C_j also ranks C_i to be the highest preferred community to join, meaning the set $L^t(C_i, k = 1)$ is not empty, they would join each other. Having set $k = 1$ is a very strict and hardly satisfiable condition. In order to relax the requirement, we increase the value of k by a rate r proportional to time slot t: $k = 1 + \lfloor r \times t \rfloor$. On early steps of the training process, web services and communities are more strict, but as time goes on, we let them choose second and then third best options too. However, increasing k increases the time complexity as well.

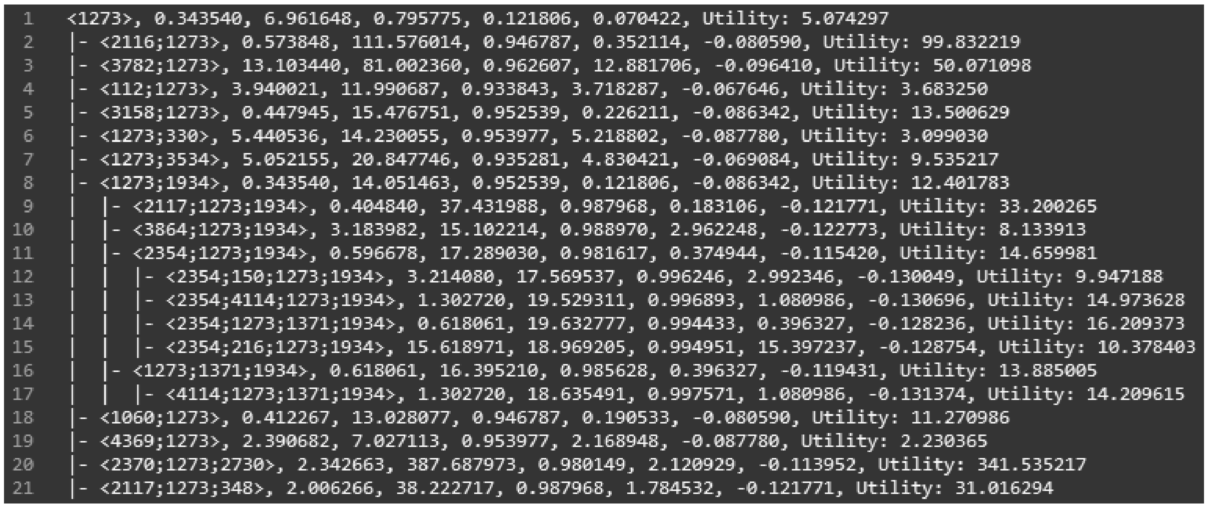
When communities C_i and C_j are in each other's top k preference set, the new combined community, i.e., $C_i \cup C_j$ is added to the list of possible communities that can join others at time $t + 1$. Moreover, for each community C_i in our initial $CFVS$ set, we maintain a tree with the community C_i as its root. Its children are all the communities that C_i decided to join. As the scenario progresses over time, the merged communities may decide to join other communities. When communities C_i and C_j decide to join each other and create community C_k, the new community C_k will be added as a child to both C_i and C_j nodes. At the end of the process, each community is utilized with a tree representing all possible combinations of communities it can join. Algorithm 1 illustrates the DDM tree creation procedure as pseudo-code.

Algorithm 1: DDM DECISION TREE ALGORITHM.

```
Input: $\langle r, gain^t_{n,n}, CFVS \rangle$ learning rate $r$, $|N \times N \times T|$ gain matrix, community
       feature vector
Output: A set of root nodes of the decision trees
1  $k \leftarrow 1$
2  $nodes[N] \leftarrow$ initialize $N$ tree nodes representing each community in CFVS
3  for $t \leftarrow 1$ to $T$ do
4      $k \leftarrow 1 + round(r \times t)$
5      for all $C_i \in CFVS$ do
6          for all $C_j \in L^t(C_i, k)$ do
7              if $C_i \in L^t(C_j, k)$ then
8                  $C_k \leftarrow C_i \cup C_j$
9                  add $C_k$ to $CFVS$ set
10                 initiate $node_k$, representing $C_k$
11                 $nodes_i.addChild(node_k)$
12                 $nodes_j.addChild(node_k)$
13 return nodes
```

Having created $|n|$ trees, one per community, our communities are utilized with the different possible paths they can take to maximize their utilities. Using a distance function[4], communities and web services outside the training set can find the community that closely resembles their parameters within the $CFVS$ set. Those new communities can use the trees of the closest communities in the training set to have an estimation of the outcome of all possible joining actions they can take. By so doing, new communities can request to join the best communities which will maximize their gain. Such a request is most likely to be accepted as the decision considers the preferences and utility gain of the other side as well.

[4]See Section 4.5 for an example of this function.



```
<1273>, 0.343540, 6.961648, 0.795775, 0.121806, 0.070422, Utility: 5.074297
 |- <2116;1273>, 0.573848, 111.576014, 0.946787, 0.352114, -0.080590, Utility: 99.832219
 |- <3782;1273>, 13.103440, 81.002360, 0.962607, 12.881706, -0.096410, Utility: 50.071098
 |- <112;1273>, 3.940021, 11.990687, 0.933843, 3.718287, -0.067646, Utility: 3.683250
 |- <3158;1273>, 0.447945, 15.476751, 0.952539, 0.226211, -0.086342, Utility: 13.500629
 |- <1273;330>, 5.440536, 14.230055, 0.953977, 5.218802, -0.087780, Utility: 3.099030
 |- <1273;3534>, 5.052155, 20.847746, 0.935281, 4.830421, -0.069084, Utility: 9.535217
 |- <1273;1934>, 0.343540, 14.051463, 0.952539, 0.121806, -0.086342, Utility: 12.401783
 |  |- <2117;1273;1934>, 0.404840, 37.431988, 0.987968, 0.183106, -0.121771, Utility: 33.200265
 |  |- <3864;1273;1934>, 3.183982, 15.102214, 0.988970, 2.962248, -0.122773, Utility: 8.133913
 |  |- <2354;1273;1934>, 0.596678, 17.289030, 0.981617, 0.374944, -0.115420, Utility: 14.659981
 |  |  |- <2354;150;1273;1934>, 3.214080, 17.569537, 0.996246, 2.992346, -0.130049, Utility: 9.947188
 |  |  |- <2354;4114;1273;1934>, 1.302720, 19.529311, 0.996893, 1.080986, -0.130696, Utility: 14.973628
 |  |  |- <2354;1273;1371;1934>, 0.618061, 19.632777, 0.994433, 0.396327, -0.128236, Utility: 16.209373
 |  |  |- <2354;216;1273;1934>, 15.618971, 18.969205, 0.994951, 15.397237, -0.128754, Utility: 10.378403
 |  |- <1273;1371;1934>, 0.618061, 16.395210, 0.985628, 0.396327, -0.119431, Utility: 13.885005
 |  |  |- <4114;1273;1371;1934>, 1.302720, 18.635491, 0.997571, 1.080986, -0.131374, Utility: 14.209615
 |- <1060;1273>, 0.412267, 13.028077, 0.946787, 0.190533, -0.080590, Utility: 11.270986
 |- <4369;1273>, 2.390682, 7.027113, 0.953977, 2.168948, -0.087780, Utility: 2.230365
 |- <2370;1273;2730>, 2.342663, 387.687973, 0.980149, 2.120929, -0.113952, Utility: 341.535217
 |- <2117;1273;348>, 2.006266, 38.222717, 0.987968, 1.784532, -0.121771, Utility: 31.016294
```

Figure 4.3: A partial view of a decision tree created by DDM.

As a real scenario example from the used data set, Figure 4.3 depicts a snapshot from a decision tree created by the DDM algorithm for a particular singleton community C_{1273}. This tree shows the different communities that C_{1273} has experienced with during the training process. Each line shows the web services list within a community, the community's feature vector and the last value on each line is the overall gained utility of the community.

Complexity. Here we analyze the computational complexity of the DDM decision tree creation algorithm on each time iteration t. Computing top k preferred communities for C_i in $K^t(C_i,k)$ requires $O(n.log(n))$ sort time. The size of $K^t(C_i,k)$ is k, and for each of those k communities, we need to check against their k top preferred communities, which needs $O(k^2)$. Line 5 iterates through n communities, Line 6 takes $O(n.log(n))$ to compute and iterates k times, and considering we already have the list sorted, Line 7 can reuse the sorted preferences. Thus, Line 7 takes $O(k)$ time to check if C_i is member of $K^t(C_i,k)$. Multiplying these iterations, the order of complexity of the algorithm with regard to n and k for each time slot is: $O(k^2 \times n^2.log(n))$. Since the whole algorithm runs T times, the overall complexity is $O(T \times k^2 \times n^2.log(n))$.

4.5 Experiments

We implemented DDM in Java[5]. We recall that we have extracted the set of features for 4532 web services in 64 different time slots through a data set provided in [90]. By randomly choosing 86 web services out of this data set for each run, and selecting a subset of all possible combinations of sizes 2, 3, and 4 of these 86 web services, we have been able to create 10,000 communities and evaluated the feature vectors and utilities they can have in the 64 time slots. This provides us with the initial training feature set of size $|CFVS| = 10,000$ communities. Based on Equation 4.9, the utilities of these communities are estimated, and then the *gain* matrix of size $|10,000 \times\ 10,000|$ of all possible ways of merging these 10,000 communities is generated[6]. Based on the *gain* matrix, each community has an ordered preference among other communities in the set.

We let communities and web services adopt their strategies based on our *DDM* decision making mechanism. Based on the decisions adopted, each community will generate a decision tree profile. We let DDM run four times with different r rates of 0.05, 0.07, 0.10 and 0.20. With the slow rate of $r = 0.05$, we increase k in Equation 4.7 for every 20 time frames, which will happen only three times in our 64-step experiment. In the case of $r = 0.20$, k increases much faster, at a rate of once every 5 time frames, which increases the complexity of the $L^t(C_i, k)$ search for each community in the $CFVS$ set.

Table 4.2 depicts the average utility gain value of the communities in each of the four runs. The utility gain is the increase of utility the communities gain by cooperating and joining other communities. The utility gain ratio is the ratio of their final utility over initial utility. Comparing the different search rates, we can see that increasing the value of r from 0.05 to 0.10 results in a significant performance boost. However, higher rates

[5]Source code of implementation and data is available at: `https://github.com/Marooned202/DDM`

[6]The template vector and *gain* matrix generated are available at `https://github.com/Marooned202/DDM/tree/master/wsds/data/run`

of r ($r > 0.10$) are not increasing the chance of finding better collaborative groups for our communities while unnecessarily increasing the search complexity.

Table 4.2: Utility gain of web services after making collaborative groups based on DDM algorithm with different r rates.

Search Rate r	Utility Gain Value	Utility Gain Ratio
r=0.20	176.1499	6.9690
r=0.10	174.6541	6.9182
r=0.07	159.9462	6.2834
r=0.05	136.0768	5.1032

The closest related work [48, 46, 41] and our previous work [8] regarding the community formation problem have considered a centralized approach where a community manager has complete information of all the web services and their quality metric and parameters. Those proposals run complex algorithms through all the space of solutions in order to find the optimal answer. However, in this research work, we have considered an unexplored and more realistic situation where information is incomplete and a decision profile is generated based on a smaller set of web services. Our solution helps communities and web services select actions that lead towards maximizing their utilities. Therefore, considering the different settings, we cannot experimentally compare our work with the mentioned related work.

To compare our work against a benchmark, we utilize the same communities and web services with a simple rational decision making mechanism in which a community will choose to join another one if it increases its utility by any amount, without aiming to be optimal. We call this method the *rational* method. We have chosen 10 random web services and compared the results with web services which adopted our DDM model. Figure 4.4 shows the comparison of the end result of utility gain values. In 18 out of 40 tries, *rational* agents were not able to improve their utility at all because the communities they chose

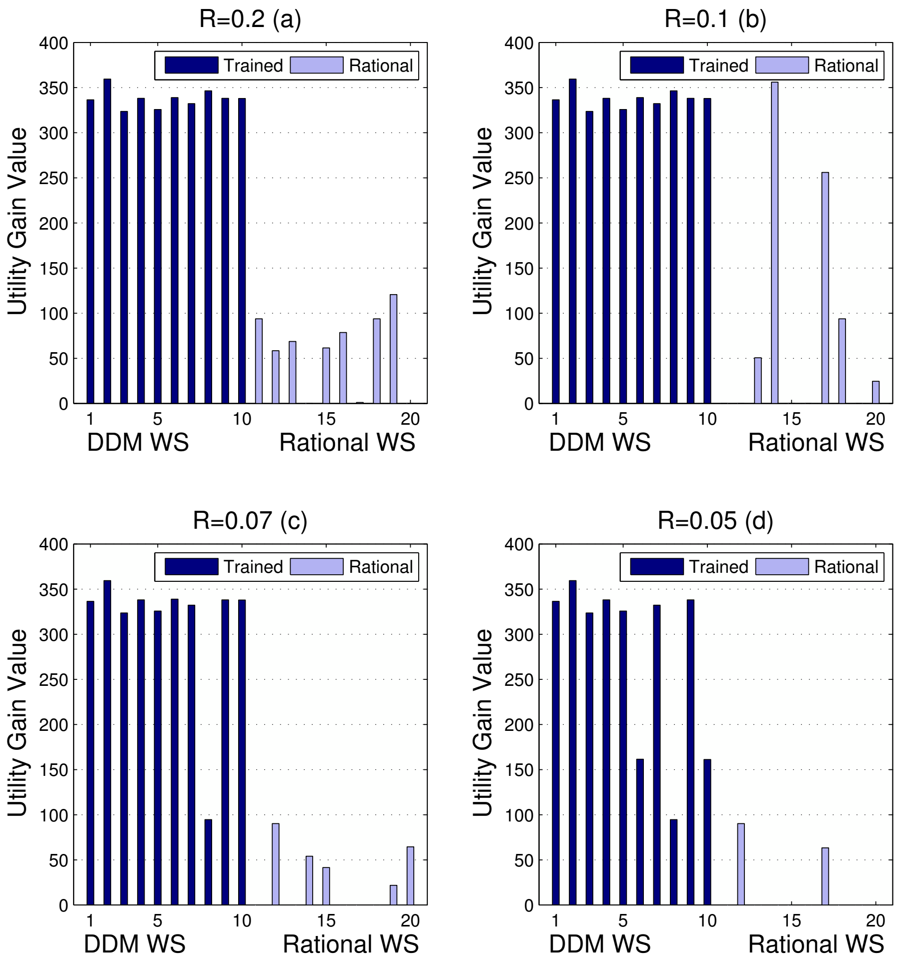


Figure 4.4: DDM against Rational: utility gain.

rejected their request, most likely because they would not have increased the utility of the other communities if they had joined them. The results show that a long-term strategic decision mechanism is needed to satisfy all the services within communities. Figure 4.5 shows the same results in terms of ratio of utility gain.

Now, we evaluate the performance of the decision profiles generated based on our data set for other communities.We create 1,000 communities from the web services in the data set that were not involved in the training process of our decision model. We define

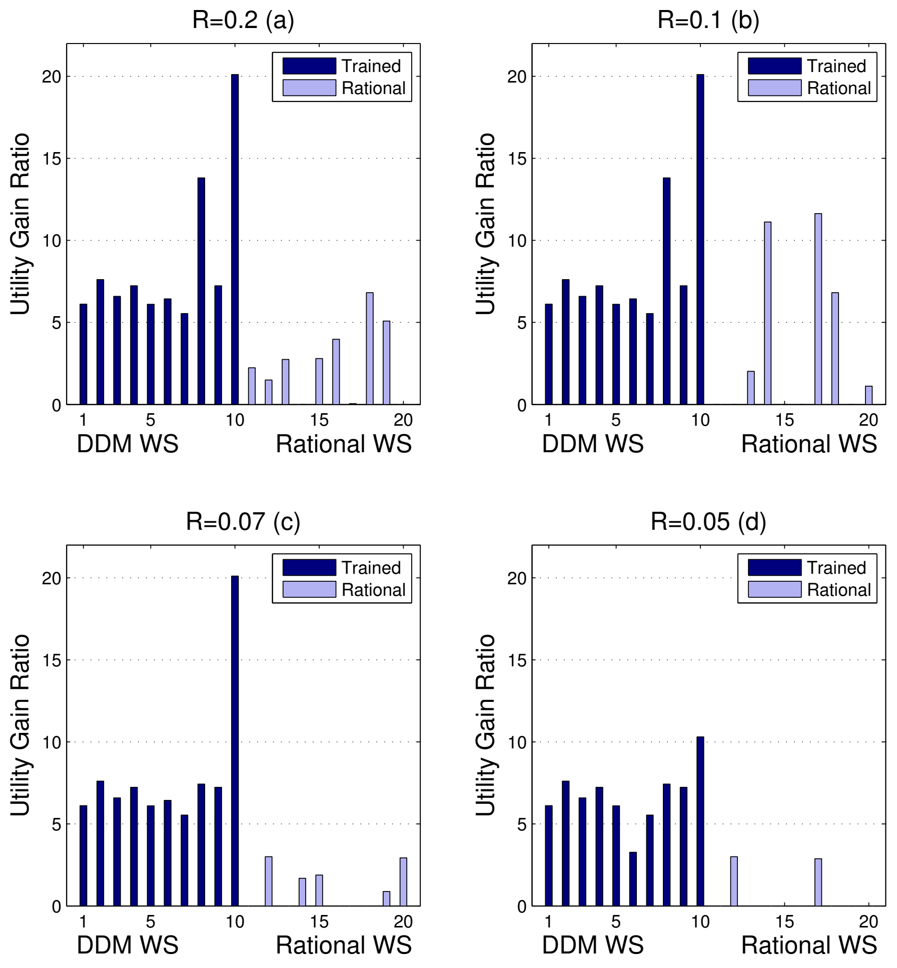


Figure 4.5: DDM against Rational: ratio of utility gain.

a distance function that measures the difference between basic features of communities, which measures the similarity among communities.

$$\begin{aligned} distance(C_1,C_2) = &|Th_{C_1} - Th_{C_2}| \\ &+ |A_{C_1} - A_{C_2}| + |Et_{C_1} - Et_{C_2}| \end{aligned} \tag{4.12}$$

Now, each community tries to find the closest community within the trained $CFVS$ set. Following its decision profile, the community can get a good estimate of the possible strategic decisions it can adopt. Basically, the trained profiles benefit the new communities in two ways. First, they provide the communities with a set of viable communities to join. Second, they provide an estimation of long-term utility gain for each available decision. In this experiment, we let communities follow the best decision within the decision tree provided to them.

In order to evaluate the performance of the mechanism, we used *Receiver Operating Characteristic (ROC) curve*, which is a graphical plot illustrating the true negative rate against the false positive rate at various threshold settings in classifier systems. In order to classify our communities' selection strategies correctly, for each community, we evaluated the training process by replacing the community in the set with the closest one, from which it gets the strategy profile. If the actions are the same and the same utility levels are gained, we classify the decision as correct. Otherwise, it is classified as a wrong decision. *AUC*, the area under the *ROC curve*, is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one, and the higher the number the better the solution, which reflects better performance. Figure 4.6 illustrates the *ROC curve* evaluation of the DDM decision making mechanism. As benchmark, we compare our method with two other methods: the *rational* method and the *greedy* method.

The *greedy* method only looks up the available list of communities and simply joins the community that maximizes its utility without considering any long-term strategy or other communities' acceptance scenarios. It is a greedy algorithm that focuses on choosing a locally optimal choice.

- **Rational Method:** Communities would send a join request to any available community, which will increase the utility. The other community would accept the join offer if its own utility gain is positive as well.
- **Greedy Method:** Communities do a linear search among all the available communities and send a join request to the community which results in maximum utility. The other community would accept the join offer if its own utility gain is positive and the utility gain does not need to be the maximum for the community receiving the join request.

Figure 4.6 compares the results for all the methods. The *rational* and *greedy* methods have very high failure rates compared to our method. Table 4.3 illustrates the number of communities that failed to find the optimal collaboration group. The results support the need for a long-term training model in a successful decision making process.

Table 4.3: Number of communities that misses the optimal decision, out of 1,000 communities.

Method	Miss
DDM r=0.05	375
DDM r=0.07	137
DDM r=0.10	6
DDM r=0.20	6
Rational Method	717
Greedy Method	828

Now, we evaluate the system-specific results from users' and communities' perspectives. By distributing tasks among the communities over the 64 time frames, we evaluate

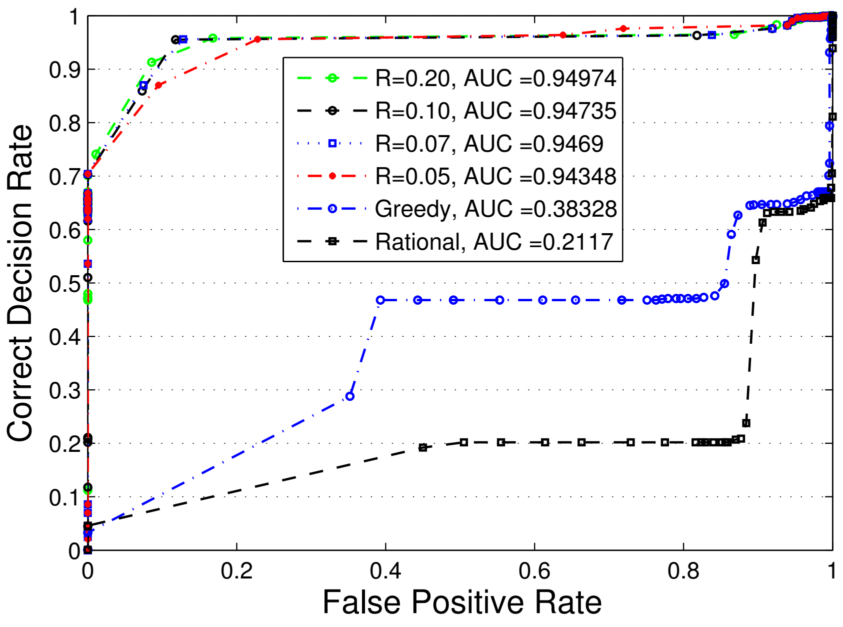


Figure 4.6: RoC Curve.

the revenue for each community. Figure 4.7 shows the overall revenue gain of communities using our method. Figure 4.8 shows the momentarily revenue gain for each community in each time slot compared to the previous time. These results show that the run with the higher learning rate of $r = 0.20$ starts discovering better communities to join much earlier. The runs with slow rates seem to find some communities to join initially, but then they slow down until later, when they start discovering new communities to join.

Figure 4.9 depicts the average community size over time, which essentially represents the number of new communities being formed. The results show once again the communities using DDM with higher search rates grow faster in size, implying that the communities find appropriate web services to join with faster.

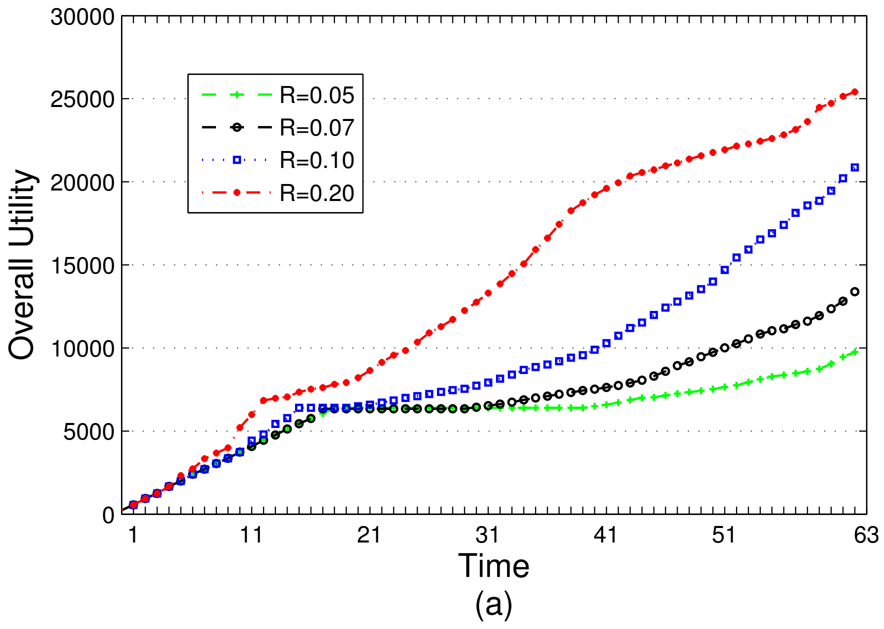


Figure 4.7: Overall utility of all the communities.

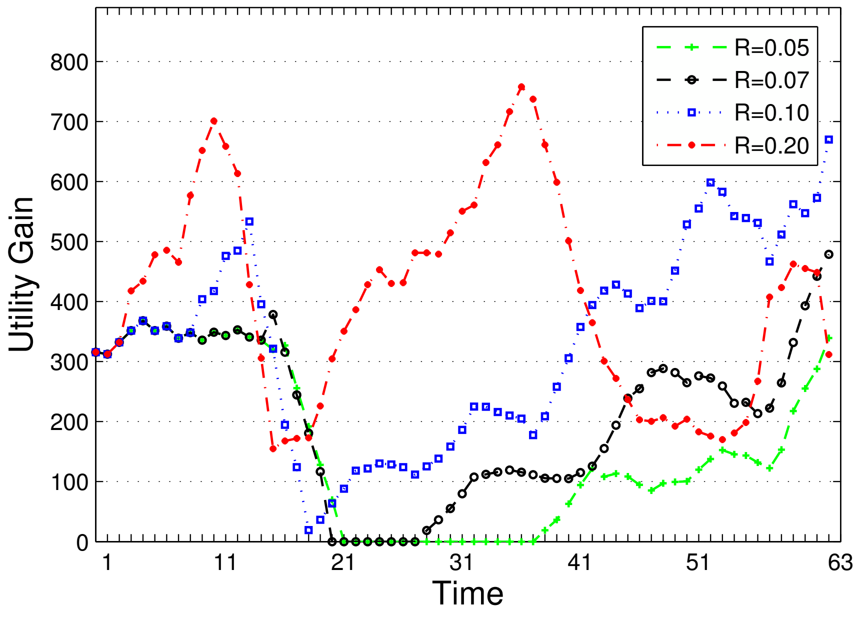


Figure 4.8: Utility gain over time.

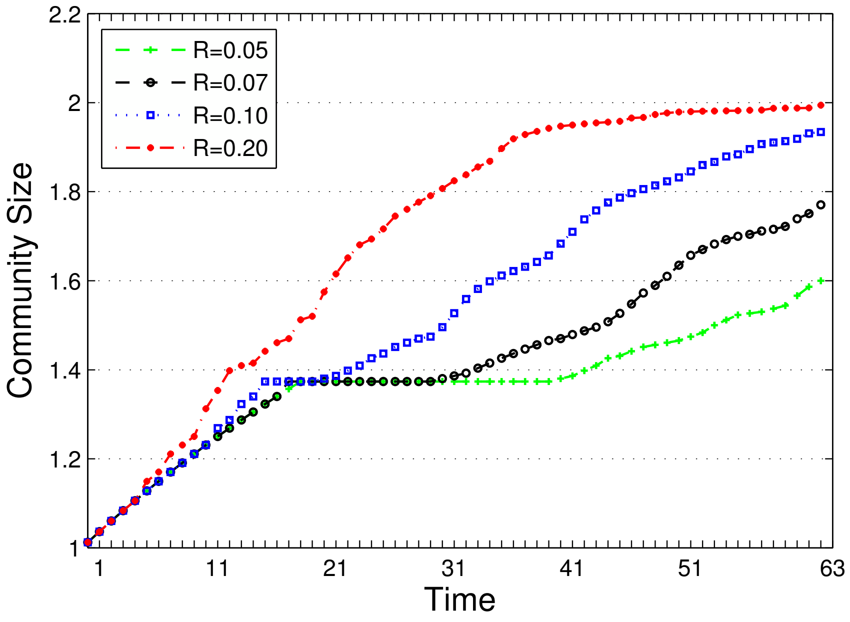


Figure 4.9: Average community size.

4.6 Summary

In this research work, we proposed a training model for the problem of membership management of communities of web services. Using the training model, we created a decision making profile for each community and web service involved which provides them with a set of feasible and utility increasing moves. This utilized our web services with efficient methods of foreseeing how their choices of actions would impact their long-term and short-term goals, which allowed them to make better decisions. The ultimate goal is to choose the best decision when it comes to communities formation, among many possible short-term rational and utility increasing choices. The experimental results show that our algorithms provide web services and community owners, in real-world-like environments, with applicable and near-perfect decision making mechanisms. The results of experiments using real data samples support the need for a long-term training model in a successful decision making process.

In this chapter, we addressed the research questions *R4* and *R5* that were introduced in section 1.3. In the next chapter, we analyse the internal community behavior of web services, and propose a model for competing and cooperating agents within the communities of web services.

Chapter 5

Coopetitive Behavior of Services within Communities

5.1 Introduction

In the previous chapter, our focus was on community formation and we emphasized cooperative behavior of the web services as agents. Within communities, the web services, selfish and utility maximizers by nature, can follow two different strategies, namely cooperation and competition in order to increase their payoffs when they provide services to consumers [50]. In typical business settings, services are used to compete within communities as they provide the same functionalities and the number of users requests is finite. However, the same reason of providing similar functionalities can lead services to cooperate because they can replace each other in case of failure or unavailability, and services can do better in a coalition structure. Analyzing services competition and cooperation strategies within communities is still an open problem that motivates the research described in this section. We propose a mechanism within which service agents in the community could choose either to compete for an announced task[1], or to cooperate with other competing services in the same community to accomplish some subtasks of the announced task. We equip intelligent web services to follow a reasoning technique to choose best interactive strategy (Coopetitive attitude, which is categorized to compete and cooperate). In the proposed system, we explore details behind the strategic decision making procedures and enable service agents to apply different techniques to constrain high efficiency and obtain the maximum utility. We investigate services' expected payoffs and the involved probabilities that are used to choose over the two interacting strategies.

Here, we first present the architecture of the proposed model. We explore the characteristics of intelligent service agents and their network. We link this architecture to the implemented system where we investigate the services' coopetitive attitudes. We compute the involved system parameters and explain the services' interactive strategy profiles by highlighting their coopetitive choices.

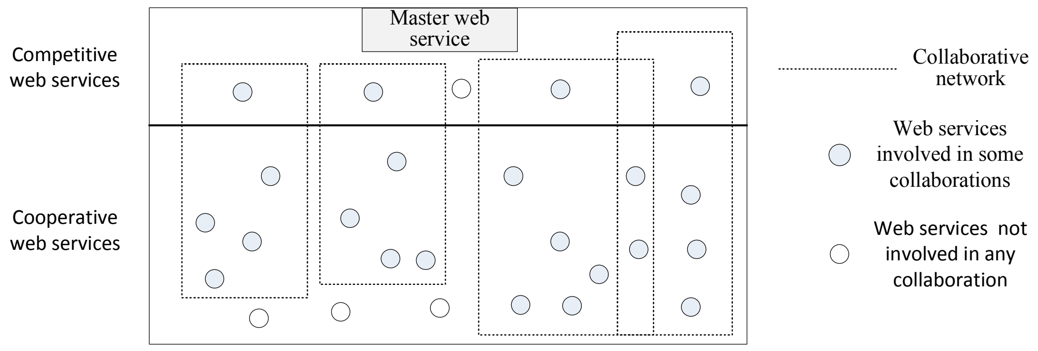

Master web service
Competitive web services
Cooperative web services
Collaborative network
Web services involved in some collaborations
Web services not involved in any collaboration

Figure 5.1: Services are partitioned into competitive and cooperative sets. Competitive services may get tasks directly from the master agent and they can share it with other cooperative services in their collaborative networks within the same community.

[1]Requests and tasks are used in this report interchangeably.

5.2 Architecture

Figure 5.1 illustrates the architecture of a typical community aggregating a number of services with different interactive strategies. Some of them compete for the task where they directly deal with the master. Some others cooperate in the associated task where they only deal with the competed service as the task leader and do not directly interact with the master (the master deals only with the service that has bid for the task, which is responsible of choosing its collaborative network). In both sets, some service agents are for certain moments out of any collaboration network. Upon allocation of the task, the service is responsible for offering the required QoS that is stated in the task being generated by a consumer. Afterwards, the master rewards or penalizes the competing service by upgrading or degrading its reputation according to the offered QoS compared with the required one. This comparison influences the sorting mechanism used by the master to allocate the tasks in further task allocation rounds.

5.3 System Parameters

In this section, we demonstrate the parameters involved and their corresponding formulations and explanations.

Task QoS (T^r_{QoS}) is the required QoS metric for a specific task r. Users define tasks with specific QoS requirements such as response time, availability, and successability (or accuracy). We aggregate and normalize these metrics to a value between 0 and 1.

Service QoS (QoS^r_w) is the QoS provided by the service w after performing the task r. Again, the metrics that contribute in computing this QoS are aggregated and normalized to a value between 0 and 1. The offered quality might or might not meet the required task quality T^r_{QoS}. In the latter case, the service user would be disappointed and a negative satisfaction feedback is expected. In our proposed system, both cases are considered when calculating the services' reputation.

Budget (B_w^t) is the amount of money the service agent w has in its disposal during the window time t (i.e., $[0,t]$), which helps pay for the community membership fees (ε) and is one of the parameters that the service agent considers when deciding about getting involved in a competition or not.

Reputation (Rep_w^t) is a significant factor in any online community [31]. Without a reputation enabling mechanism, users cannot differentiate among services, specially the ones which offer the same type of service. Reputation mechanisms usually aggregate users' experiences and in our case it strongly depends on QoS that each service provides. Users define tasks, each one with specific quality T_{QoS}^r, so that after performing a certain number of tasks, each one with QoS_w^r, during a window time t, the reputation of w gets evaluated by the master agent. Rep_w^t refers to the reputation of w during that window time t.

In Equation 5.1, we compute the reward that the master computes considering the task r's QoS T_{QoS}^r compared with the service offered quality QoS_w^r. In case the offered quality meets user expectations, the reward value would be positive. In this system, we consider a small value as default rewards η which the master considers together with the proportional level of satisfaction as a weighted value (by υ). In this case, the higher the offered quality, the more weighted reward. In case the offered quality did not meet the user expectations, the reward would be negative. A default penalty value ρ (where $\rho > \eta$) together with the weighted proportional difference are therefore considered. The idea is to harshly penalize the services rather than rewarding them. To this end, rational service agents should carefully analyze their capabilities once the available tasks are announced. Equation 5.2 computes the obtained reward by w during the window time t considering the set $task_w^t$ of tasks performed by w during the window time t. In our proposal, service agents have the goal of increasing their budget, which is directly related to their reputation. Thus, they have to decide strategically how to maximize this value.

$$reward_w^r = \begin{cases} \eta + \upsilon \frac{QoS_w^r}{T_{QoS}^r + QoS_w^r} & \text{if } T_{QoS}^r \leq QoS_w^r; \\ -(\rho + \upsilon \frac{T_{QoS}^r}{T_{QoS}^r + QoS_w^r}) & \text{otherwise.} \end{cases} \tag{5.1}$$

$$reward_w^t = \begin{cases} \frac{\sum_{r \in task_w^t} reward_w^r}{|task_w^t|} & \text{if } task_w^t \neq \emptyset; \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

The assigned reputation value is updated by the computed reward value. The computed reputation of services is bounded by the minimum and maximum reputation values 0 and 1. Let $\Gamma = Rep_w^t + reward_w^t$. The updated reputation value is then computed as follows:

$$Rep_w^{t+1} = \begin{cases} \Gamma & \text{if } 0 \leq \Gamma \leq 1; \\ 0 & \text{if } \Gamma < 0; \\ 1 & \text{if } \Gamma > 1. \end{cases} \tag{5.3}$$

For new services with no previous reputation value, we use the bootstrapping trust technique proposed in [87]. This technique consists in giving the new services a chance and observe their behaviors for a period of testing time. The observation sequence is modeled as a hidden Markov model that is used to detect the behavior of the service by comparing the observation behavior against pre-defined trust patterns. Based on the matching result, an initial value is assigned to the service. Using this initial reputation value, services quickly converge to their actual and stable values using the update function.

Proposition 5.1. *Rep_w^t can be computed in time $O(|t|)$, i.e., in time linear in the size of the window t.*

Proof. [2] The function Rep_w^t is recursive on t, but the algorithm works by storing the last calculated reputation value in a variable, so it will not be recalculated again at each iteration. However, the calculation of $reward_w^t$ is needed. Since the function $reward_w^t$ can be computed in time linear in the number of tasks (see Equations 5.1 and 5.2), which in turn is linear in the size of the window time, the result follows. □ □

Growth Factor (G_w^t) is a parameter which declares services' performance based on their recent strategies and activities. Growth factor is relative to services' reputation Rep_w^t, QoS during the window time t QoS_w^t, and budget B_w^t. This factor is the main variable a typical service uses to decide about which strategy to adopt. We use Equation 5.4 to compute the growth factor G_w^t of the service w during the window time t as the average of the three aforementioned parameters, where n_t is the total number of offered tasks to the whole community during the window time t, μ_w is the mean received service fee, and ε is the cost of community membership.

$$G_w^t = \frac{Rep_w^t + QoS_w^t + \frac{B_w^t}{n_t \mu_w - \varepsilon}}{3} \tag{5.4}$$

$$\mu_w \in \{\mu_{w,CM}, \mu_{w,CO}\}, \quad QoS_w^t = \begin{cases} \frac{\sum_{r \in task_w^t} QoS_w^r}{|task_w^t|} & \text{if } task_w^t \neq \emptyset; \\ 0 & \text{otherwise.} \end{cases}$$

This equation is designed so that it satisfies the following desirable properties:

1. The growth factor function should be monotonically increasing in the offered quality of service QoS_w^t.

2. The growth factor function should be monotonically increasing in the service's reputation Rep_w^t.

[2] In this work, we assume that the common arithmetic and elementary functions, such as multiplication, division and trigonometric functions can be computed in time $O(1)$ as they operate on inputs of fixed sizes.

3. The growth factor function should be monotonically increasing in the budget B_w^t if the maximum possible profit is positive and monotonically decreasing in B_w^t if the maximum possible profit is negative. This property reflects the idea that the budget contributes in the increase of the growth factor as far as there is a chance to make profit. In fact, the contribution of the budget B_w^t in the calculation of the growth factor should be proportional to the maximum possible profit $n_t\mu_w - \varepsilon$ where the service w receives all the offered tasks during the window time t.

It is easy to show that Equation 5.4 satisfies the three aforementioned properties by calculating the partial derivatives ∂G_w^t of this function in 1) QoS_w^t ($\frac{\partial G_w^t}{\partial QoS_w^t} = \frac{1}{3}$); 2) Rep_w^t ($\frac{\partial G_w^t}{\partial Rep_w^t} = \frac{1}{3}$); and 3) B_w^t ($\frac{\partial G_w^t}{\partial B_w^t} = \frac{1}{3(n_t\mu_w - \varepsilon)}$). Thus, the sign of the two first partial derivatives is positive and the sign of $\frac{\partial G_w^t}{\partial B_w^t}$ depends on the sign of the maximum profit $n_t\mu_w - \varepsilon$, so we are done. The mean service fee depends on the strategy adopted by the service because a competitive service receives higher fees $\mu_{w,CM}$ compared to a cooperative one $\mu_{w,CO}$ ($\mu_{w,CM} > \mu_{w,CO}$). The motivation behind this is that a competitive service for a given task is the leader for that task while other cooperative services are performing specific subtasks as asked by the leader.

Proposition 5.2. *G_w^t can be computed in time linear in the size of the window t.*

Proof. As shown in the second part of Equation 5.4, the function QoS_w^t can be computed in time linear in the number of tasks, which in turn is linear in the size of the window time. Since B_w^t is constant, the result follows from Proposition 5.1. □ □

The above explained parameters and other additional parameters which will be used in the rest of this chapter are listed and self explained in Table 5.1.

Table 5.1: List of abbreviations.

Notation	Definition	Notation	Definition
T^r_{QoS}	Required task QoS for the task r	T^t_{QoS}	Mean required task QoS during t
QoS^r_w	Service w QoS for the task r	QoS^t_w	w QoS during the window time t
$Reward^r_w$	Reward obtained by w w.r.t. r	$Reward^t_w$	Reward to update the reputation
Rep^t_w	Reputation assigned for w	B^t_w	Budget associated to service w
G^t_w	Growth factor of w during t	ε	Community membership fee
$\pi^t_{w,CM}$	Competition payoff of w	$\pi^t_{w,CO}$	Cooperation payoff of w
$p^t_{w,CM}$	Competition probability of w	$p^t_{w,CO}$	Cooperation probability of w
COF^t_w	Cooperation fee of w	β_w	Profit of w
$\mu_{w,CM}$	Mean service fee for competing w	$\mu_{w,CO}$	Mean service fee for cooperating w
τ^t_w	Coopetitive threshold of w	P^t_w	Probability of competing for w
U^t_w	Utility of w	E^t_w	Expected number of tasks

5.4 Service Interactive Strategies

The main goal of each individual service agent is to increase its income (payoff). This income can be earned from tasks (or requests) done by this service. In our model, services can decide to compete to get a task from the master agent or to cooperate with other services within a given collaborative network (the way a collaborative network is set by a leader is based on the cooperative services reputation and their QoS parameters that should coincide with the required QoS). Therefore we define two types of service strategies. First, when a service has higher level of confidence based on its growth factor, it can compete to get a task from the master and adopts the competitive strategy. Second, when the service agent has a lower level of confidence that it does not feel capable to compete, it waits for some other services to cooperate with to perform some tasks [3], and thus it adopts the cooperative strategy. Services estimate the outcome of all the strategies and choose one of them accordingly. This decision is not static but can change over time so service agents can switch from one strategy to the other, and this dynamic attitude is referred to as coopetition. The underlying decision making process is presented in the next section.

[3]Through the report, requests or tasks are supposed to be decomposable.

5.5 Theoretical Results

5.5.1 Service Decision Making Procedure

In this section, we explore in details the interaction strategies and the outcome of each strategy in terms of services' utilities. The main part of services' decision making procedure falls into their growth factor analysis. In fact, the comparison of the growth factor to a particular threshold is the main reason that influences the service's decision to follow either competitive or cooperative behavior. service agents initially compute this value and compare it with their computed threshold. Generally the main challenge is the threshold computation and we cope with this issue in the rest of this section. We additionally use the obtained results in the implemented environment and analyze their effectiveness on services' strategic decision making procedures.

Figure 5.2 shows the decision making process that is followed by a typical service. In case the service agent is ready to compete, there is a chance that it bids for a task if it has the required capabilities to accomplish that task, or stays silent and returns to the cooperative status. But in case the service agent is willing to cooperate, it has to wait for a cooperation opportunity that could be triggered by another service agent that competed and obtained the task, so both services will be part of the same collaborative network. In the decision making process presented in Figure 5.2, we assume that the competing service might get the task (denoted as $Bid/obtainedTask$) or not in case of being rejected by the master agent, or do not even bid for the task (denoted as $Silent/rejectedTask$). For simplicity reasons and without loss of generality, we group the two cases of $Bids$ and $obtainedTask$ together as well as $Silent$ and $rejectedTask$. The rational behind this aggregation is the fact that our main concentration is services' status (competitive or cooperative) over different decision making rounds, which could be caused by internal factors (the services) or by the external factor (the master agent).

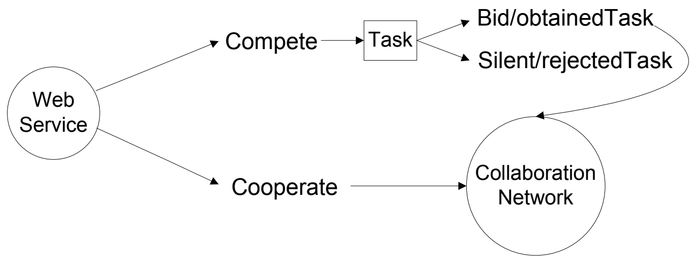


Figure 5.2: Decision making process over competitive and cooperative strategies.

Consider a service w that is willing to compete for the period of time t (that means the computed growth factor is more than the analyzed threshold). This service can estimate the expected payoff associated to this decision, called *competition payoff*. Equation 5.5 computes this expected payoff for the competing service w ($\pi^t_{w,CM}$) considering the $Bid/obtainedTask$ probability of $p^t_{w,CM}$ and $Silent/rejectedTask$ probability of $1 - p^t_{w,CM}$.

$$\pi^t_{w,CM} = p^t_{w,CM}(\mu_{w,CM}E^t_w - COF^t_w E^t_w - \varepsilon) + (1 - p^t_{w,CM})(-\varepsilon) \tag{5.5}$$

In Equation 5.5, E^t_w is the number of tasks that w expects during the window time t, and $\mu_{w,CM}$ is the mean service fee that is assigned by the master agent to w. This means that a competing service directly obtains this fee from the master agent. Moreover, the competing service w expects a cooperation fee (COF^t_w) that it gives to its collaborators in case w needs to cooperate with other services (cooperative service agents in its collaboration network). In any case, the competing or cooperating service agent pays a fixed amount of membership (ε) to the master agent, which plays the role of the community's coordinator. This fee would be taken into account when a service decides to leave to a cheaper community or act alone. But to concentrate on the main concerns of this thesis, we skip these small details.

From Equation 5.5, the following proposition is straightforward.

Proposition 5.3. *The complexity of computing the competition payoff $\pi^t_{w,CM}$ is linear in the competition probability $p^t_{w,CM}$, the expected number of tasks E^t_w, and the cooperation fee COF^t_w.*

The arrival of requests for service w during the time unit t (denoted here by $m_w(t)$) can be modeled as a nonhomogeneous Poisson process [42, 73], which means as a Poisson process with dynamic arrival rate $\lambda_w(x)$ where x belongs to the time unit t. The arrival rate is thus a function of time and typically varies significantly from moment to moment. In nonhomogeneous Poisson process, $m_w(t)$ is expressed as follows:

$$m_w(t) = \int_0^t \lambda_w(x)\, \mathrm{d}x$$

And the probability of having exactly n requests during the window t is computed as follows:

$$p(m_w(t) = n) = \frac{(m_w(t))^n}{n!\; e^{m_w(t)}}$$

Let Max^t_w be the maximum number of requests that w can receive during t. The number of expected requests E^t_w is given by the parameter $\lambda_w(t)$ as follows:

$$E^t_w = \lambda_w(t) = \sum_{n=1}^{Max^t_w} n\; p(m_w(t) = n) \tag{5.6}$$

The parameter $\lambda_w(t)$ is usually estimated from data samples using the least squares, iterative least squares, or maximum likelihood [57].

Proposition 5.4. *The complexity of computing the expected number of requests E^t_w is linear in the size of the window time t.*

Proof. For a fixed function $\lambda_w(x)$, $m_w(t)$ can be computed in $O(1)$. Thus, from Equation 5.6, it follows that E_w^t can be computed in time linear in Max_w^t. As Max_w^t is linear in t, the result follows. □ □

Similar to the competitive service case, if a service w declares cooperative status, its expected *cooperation payoff* ($\pi_{w,CO}^t$) is computed in Equation 5.7. In this equation, $p_{w,CO}^t$ is the probability of getting involved in a cooperative task with other services and $1 - p_{w,CO}^t$ is the probability of failure to find such a cooperation opportunity. These two probabilities are set when w decides to compete. We recall that $\mu_{w,CO}$ denotes the mean cooperation fee that is directly obtained from the leader (i.e., the competitive) service of the underlying collaborative network. Compared to $\mu_{w,CM}$, $\mu_{w,CO}$ is relatively smaller since the competitive service generally dedicates a portion of its obtained income to pay other cooperative services.

$$\pi_{w,CO}^t = p_{w,CO}^t(\mu_{w,CO}E_w^t - \varepsilon) + (1 - p_{w,CO}^t)(-\varepsilon) \tag{5.7}$$

From Equation 5.7, the following proposition holds.

Proposition 5.5. *The complexity of computing the cooperation payoff $\pi_{w,CO}^t$ is linear in the cooperation probability $p_{w,CO}^t$ and the expected number of tasks E_w^t.*

To analyze the expected payoffs obtained from different strategies, services need to compute the estimated probabilities that distinguish subcases in each behaviorial status ($p_{w,CM}^t$ for competitive and $p_{w,CO}^t$ for cooperative). To estimate these probabilities, we should notice that they are functions of services' reputation values (Rep_w^t). Furthermore, $p_{w,CM}^t$ is also function of the difference between the offered QoS (QoS_w^t) and the mean requested one considering the set of all tasks $task^t$ (T_{QoS}^t (see Equation 5.8)); and $p_{w,CO}^t$ is function of the reputation of other services in the community because the leader is supposed to be selective when it comes to choose the collaborators. To this end, we first discuss the desirable properties of an estimation function of each of these probabilities, and show that the proposed ones satisfy those properties.

$$T^t_{QoS} = \begin{cases} \frac{\sum_{r \in task^t} T^r_{QoS}}{|task^t|} & \text{if } task^t \neq \emptyset; \\ 0 & \text{otherwise.} \end{cases} \tag{5.8}$$

Proposition 5.6. *T^t_{QoS} can be computed in time linear in the size of the window t.*

Proof. From Equation 5.8, T^t_{QoS} can be computed in time linear in $|task^t|$, which in turn is linear in the window size $|t|$. □

□

The desired properties of $p^t_{w,CM}$ are as follows:

Property 5.1. *$p^t_{w,CM}$ is continuous with regard to Rep^t_w, QoS^t_w, and T^t_{QoS}.*

Property 5.2. *$p^t_{w,CM}$ is monotonically increasing in Rep^t_w and $QoS^t_w - T^t_{QoS}$ while $QoS^t_w - T^t_{QoS}$ is positive.*

Property 5.3. *$p^t_{w,CM}$ is null if $QoS^t_w - T^t_{QoS}$ is negative.*

Property 5.4. *The increase slope of $p^t_{w,CM}$ is higher when the reputation Rep^t_w increases in the interval $[0,0.5]$ than when it increases in the interval $]0.5,1]$.*

Property 1 simply says that the probability of success competition $p^t_{w,CM}$ can be always estimated as far as Rep^t_w, QoS^t_w, and T^t_{QoS} are available. Property 2 says that the reputation and QoS are two key factors that influence the value of $p^t_{w,CM}$ in the sense of positive correlation. Property 3 indicates that the probability $p^t_{w,CM}$ is null if the offered QoS is less than the expectation. Property 4 promotes the increase of the reputation for new comers and imposes higher increase rate at the beginning of the reputation curve because it is always hard to build the reputation, but once it is built, its maintenance is less challenging.

The desired properties of $p^t_{w,CO}$ are as follows:

Property 5.5. *$p^t_{w,CO}$ is continuous with regard to Rep^t_w and the reputation of other services in the community.*

Property 5.6. *$p^t_{w,CO}$ is monotonically increasing in Rep^t_w and monotonically decreasing in the community average reputation.*

Property 5.7. *The increase slope of $p^t_{w,CO}$ is higher when the reputation Rep^t_w increases in the interval $[0, 0.5]$ than when it increases in the interval $]0.5, 1]$.*

Property 5 is similar to Property 1. Property 6 says that w has more chance to get involved in a cooperation if it has high reputation compared to the other members. This chance decreases if other services have higher reputation. Property 7 is similar to Property 4.

Equations 5.9 and 5.10 respectively compute the estimated success probability in cases where service w is competing and cooperating. These values are computed considering service's reputation value (Rep^t_w computed by the master), service's offered QoS (QoS^t_w), the task required QoS (T^t_{QoS}), which is the mean required QoS computed from previous tasks, the maximum offered QoS (QoS^t_k, which is provided by another competitive service k), and the cooperative factor CL^t_w of service w during the window time t, which is computed as the portion of service's current reputation on the average reputation of the community $\mathscr{C}$.

$$p^t_{w,CM} = \begin{cases} \sin(Rep^t_w \frac{\pi}{2}) \frac{QoS^t_w - T^t_{QoS}}{Max_k(Qos^t_k - T^t_{QoS})} & \text{if } QoS^t_w \geq T^t_{QoS}; \\ 0 & \text{otherwise.} \end{cases} \tag{5.9}$$

$$p^t_{w,CO} = \sin(Rep^t_w \frac{\pi}{2}) CL^t_w \tag{5.10}$$

$$CL_w^t = \frac{Rep_w^t}{\sum_{k\in\mathscr{C}} Rep_k^t/|\mathscr{C}|}$$

Theorem 5.1. *Equation 5.9 satisfies Properties 1 to 4.*

Proof. It is easy to show the continuity of Equation 5.9, which satisfies Property 1. The partial derivative $\frac{\partial p_{w,CM}^t}{\partial Rep_w^t} = \frac{\pi}{2}\cos(Rep_w^t \frac{\pi}{2})\frac{QoS_w^t - T_{QoS}^t}{Max_k(Qos_k^t - T_{QoS}^t)}$ is positive as the function cos (the derivative of sin) is positive on $[0, \frac{\pi}{2}]$, $Rep_w^t \in [0,1]$, and $QoS_w^t \geq T_{QoS}^t$. The partial derivative $\partial p_{w,CM}^t$ with regard to $QoS_w^t - T_{QoS}^t$ ($\frac{\partial p_{w,CM}^t}{\partial(QoS_w^t - T_{QoS}^t)} = \frac{\sin(Rep_w^t \frac{\pi}{2})}{Max_k(Qos_k^t - T_{QoS}^t)}$) is also positive since $Qos_k^t - T_{QoS}^t > 0$ and $Rep_w^t \frac{\pi}{2} \in [0, \frac{\pi}{2}]$ and sin is positive on $[0, \frac{\pi}{2}]$, which proves the satisfaction of Property 2. Property 3 is straightforward. Finally, the increase slope of the function sin on $[0, \frac{\pi}{2}]$ proves Property 4. □ □

Theorem 5.2. *Equation 5.10 satisfies Properties 5 to 7.*

Proof. We can easily show the continuity of Equation 5.10 from which Property 5 follows. Property 6 can be shown by calculating the partial derivative $\partial p_{w,CO}^t$ first with regard to Rep_w^t and second with regard to the community $\mathscr{C}$ average reputation $\sum_{k\in\mathscr{C}} Rep_k^t/|\mathscr{C}|$, where $|\mathscr{C}|$ is the cardinality of the considered community $\mathscr{C}$. The first partial derivative ($\frac{\pi}{2}\cos(Rep_w^t \frac{\pi}{2})CL_w^t$) is positive and the second ($\frac{-\sin(Rep_w^t \frac{\pi}{2})Rep_w^t}{(\sum_{k\in\mathscr{C}} Rep_k^t/|\mathscr{C}|)^2}$) is negative, which proves the satisfaction of Property 6. The proof of satisfaction of Property 7 is similar to the one of Property 4. □ □

Proposition 5.7. *$p_{w,CM}^t$ can be computed in time linear in the window size $|t|$.*

Proof. The result follows directly from 1) Equation 5.9; 2) Proposition 5.1 (Complexity of Rep_w^t is linear in the window size $|t|$); 3) second part of Equation 5.4 (the function QoS_w^t can be computed in time linear in the number of tasks, which in turn is linear in the size of the window time); 4) Proposition 5.6 (Complexity of T_{QoS}^t is linear in the size of the window t); and 5) the fact that those functions are computed independently one from the other. □ □

Proposition 5.8. *$p^t_{w,CO}$ can be computed in time $O(|t|.|\mathscr{C}|)$, which means linear in both the size of the window t and the size of the community $\mathscr{C}$.*

Proof. From Proposition 5.1, Rep^t_w can be computed in $O(|t|)$. Consequently, the function $\sum_{k\in\mathscr{C}} Rep^t_k$ can be computed in $O(|t|.|\mathscr{C}|)$, so it does the computation of the function CL^t_w as Rep^t_w will be computed just once and stored in a variable. The same variable will be used to compute $\sin(Rep^t_w\frac{\pi}{2})$. Thus, from Equation 5.10, the result follows. □ □

5.5.2 Coopetition Threshold

In this part, we compute the coopetition threshold that a typical service agent could use to adopt reasonable interacting strategies and we empirically verify the effectiveness of the obtained results in the next section. In fact, to decide which strategy to adopt, we let the service agent w compare its growth factor G^t_w with the coopetition threshold τ^t_w it holds at current window time t and choose to compete with probability P^t_w that we compute in Equation 5.11. Based on this probability, we calculate the total utility U^t_w in Equation 5.12.

$$P^t_w = \begin{cases} \frac{G^t_w}{\tau^t_w} & \text{if } G^t_w \leq \tau^t_w; \\ 1 & \text{otherwise.} \end{cases} \tag{5.11}$$

$$U^t_w = P^t_w(\pi^t_{w,CM}) + (1 - P^t_w)(\pi^t_{w,CO}) \tag{5.12}$$

The key factor in the computation of the probability P^t_w and the associated utility is the threshold value. To compute the threshold, we use the game-theoretic best response technique. A typical service agent w will follow the best response strategy to maximize its expected aggregated payoff. The idea is to equalize the expected payoffs of the two acting strategies: compete and cooperate. The objective behind equalizing payoffs is to explore conditions under which service agent w could react with best response to further decision making procedures. We use the obtained conditions to compute the threshold τ_w^t during the window time t. By equalizing $\pi_{w,CM}^t$ and $\pi_{w,CO}^t$, we obtain:

$$\pi_{w,CM}^t = \pi_{w,CO}^t \rightarrow$$

$$p_{w,CM}^t(\mu_{w,CM} - COF_w^t - \varepsilon) + (1 - p_{w,CM}^t)(-\varepsilon) = p_{w,CO}^t(\mu_{w,CO} - \varepsilon) + (1 - p_{w,CO}^t)(-\varepsilon)$$

Which means:

$$p_{w,CM}^t(\mu_{w,CM} - COF_w^t - \varepsilon) = p_{w,CO}^t(\mu_{w,CO} - \varepsilon) + (-\varepsilon)(p_{w,CM}^t - p_{w,CO}^t)$$

So, we obtain:

$$\mu_{w,CM} - COF_w^t - \varepsilon = \frac{p_{w,CO}^t \ \mu_{w,CO}}{p_{w,CM}^t} - \varepsilon$$

Therefore:

$$\mu_{w,CM} - COF_w^t = \frac{p_{w,CO}^t \ \mu_{w,CO}}{p_{w,CM}^t}$$

From which, we derive:

$$COF_w^t = \mu_{w,CM} - \frac{p_{w,CO}^t \ \mu_{w,CO}}{p_{w,CM}^t}$$

Replacing $p_{w,CM}^t$ and $p_{w,CO}^t$ using Equations 5.9 and 5.10, we derive the following:

$$COF_w^t = \mu_{w,CM} - \frac{sin(Rep_w^t \frac{\pi}{2}) CL_w^t \ \ \mu_{w,CO}}{sin(Rep_w^t \frac{\pi}{2}) \frac{QoS_w^t - T_{QoS}^t}{Max_k(Qos_k - T_{QoS}^t)}}$$

By simplifying the *sinus* function from both the numerator and denominator sides and substituting the cooperation factor CL_w^t of service w we obtain Equation 5.13:

$$COF_w^t = \mu_{w,CM} - \frac{Rep_w^t \ |\mathscr{C}|}{\sum_{k \in \mathscr{C}} Rep_k^t} \frac{\mu_{w,CO} Max_k(Qos_k^t - T_{QoS}^t)}{QoS_w^t - T_{QoS}^t} \tag{5.13}$$

Equation 5.13 computes the cooperation fee COF_w^t that is assigned by service w. This fee represents the amount that w spends to cooperate with other service(s) to accomplish the task. By so doing, we obtain the maximum amount of cooperation fee that service w can use to constrain the positive payoff out of competing. Otherwise, the service stays as cooperative entity.

Proposition 5.9. *COF_w^t can be computed in time $O(|t|.|\mathscr{C}|)$, which means linear in both the size of the window t and the size of the community $\mathscr{C}$.*

Proof. From Proposition 5.1, Rep_w^t can be computed in $O(|t|)$. Consequently, the function $\sum_{k\in\mathscr{C}} Rep_k^t$ can be computed in $O(|t|.|\mathscr{C}|)$. Since QoS_w^t and T_{QoS}^t can be computed in time $O(t)$ (from the second part of Equation 5.4 and Proposition 5.6 respectively), we are done.
□ □

Lemma 5.1. *The competition payoff $\pi_{w,CM}^t$ can be computed in time $O(|t|.|\mathscr{C}|)$.*

Proof. The result follows directly from Propositions 5.3, 5.4, 5.7, and 5.9. □ □

Lemma 5.2. *The cooperation payoff $\pi_{w,CO}^t$ can be computed in time $O(|t|.|\mathscr{C}|)$.*

Proof. The result follows directly from Propositions 5.4, 5.5, and 5.8. □ □

We use the maximum cooperation fee that a service agent considers to constrain the positive expected payoff when the competitive strategy is adopted to update the threshold for the consequent time window $(t+1)$. We compare the maximum cooperation fee with the required fee ($ReqF_w^t$) that the service indicates to accomplish the task. The outcome of this comparison is a factor that uses the current threshold τ_w^t to compute the consequent threshold τ_w^{t+1}. As in online learning, the idea is to compute iteratively the threshold until the fixed point is achieved, which indicates the threshold's conversion, where the initial value is randomly chosen (in the simulation different initial values are used). Equation 5.14

shows this computation. To investigate the effectiveness of this threshold on the outcomes of the services that follow this reasoning technique, in the next section, we compare the results of different agents with diverse strategic reasoning techniques.

$$\tau_w^{t+1} = \begin{cases} \Theta & \text{if } 0 \leq \Theta \leq 1 \\ 1 & \text{if } \Theta > 1; \\ 0 & \text{if } \Theta < 0. \end{cases} \tag{5.14}$$

$$\Theta = \tau_w^t \, \frac{COF_w^t}{ReqF_w^t}$$

Proposition 5.10. *The threshold τ_w^t can be computed in time $O(|t|.|\mathscr{C}|)$, which means linear in both the size of the window t and the size of the community $\mathscr{C}$.*

Proof. From Equation 5.14, the computation of τ_w^t is recursive on t, and the algorithm works by keeping the last computed value in a variable, which saves the time of re-calculation. Thus, the complexity of calculating τ_w^t is determined by the complexity of calculating COF_w^t since $ReqF_w^t$ is constant during the period t. Consequently, the result follows from Proposition 5.9. □ □

Theorem 5.3. *The time complexity of the proposed decision mechanism is $O(|t|.|\mathscr{C}|)$, which means linear in both the size of the window t and the size of the community $\mathscr{C}$.*

Proof. The procedure mechanism is based on comparing the growth factor G_w^t with the coopetition threshold τ_w^t as shown in Equations 5.11 and 5.12. Thus, the result follows from Propositions 5.2 and 5.10. □ □

5.6 Experimental Results and Analysis

In this section, we provide an empirical analysis over the theoretical results regarding the characteristics of intelligent service agents hosted in different communities of services. In the implemented system, we simulate the behaviors of service consumers as request generators, service agents as service providers, and master agents as community representatives. The objective is to investigate the effectiveness of the proposed strategic system on intelligent services' overall budget and also the average quality and quantity of tasks performed by the community of services, which directly affects user satisfaction. To verify these objectives, we study the overall performance of the community hosting the reasoning-empowered services compared to the ones hosting stochastic and purely competitive services. By stochastic services, we mean services that adopt at each moment competitive or cooperative strategies in an equally but random way. By equally, we mean the choices are fairly divided between the two strategies.

The simulation application is written in *C#* using *Visual Studio*. We performed the implementation on a single Intel Xeon X3450 machine with 6GBs of memory. Web services were modeled as a *class* and using *Await* and *Async* models we initiated many web services, each running as a thread. We implemented XML based messaging system (like SOAP) with request parameters and a list of XML based responses. The request contains the flight dates, the origin and destination, type of tickets, and number of guests. The response contains different flights with different companies, prices, timing, etc. A pool of services are initialized with values taken from a real dataset that includes 2507 real services functioning on the web. The dataset records the QoS values of 9 parameters including *availability*, *throughput*, and *reliability* [1].

We start our discussions with cumulative budget comparison regarding different communities within which services follow different reasoning techniques. Figure 5.3 part (a)

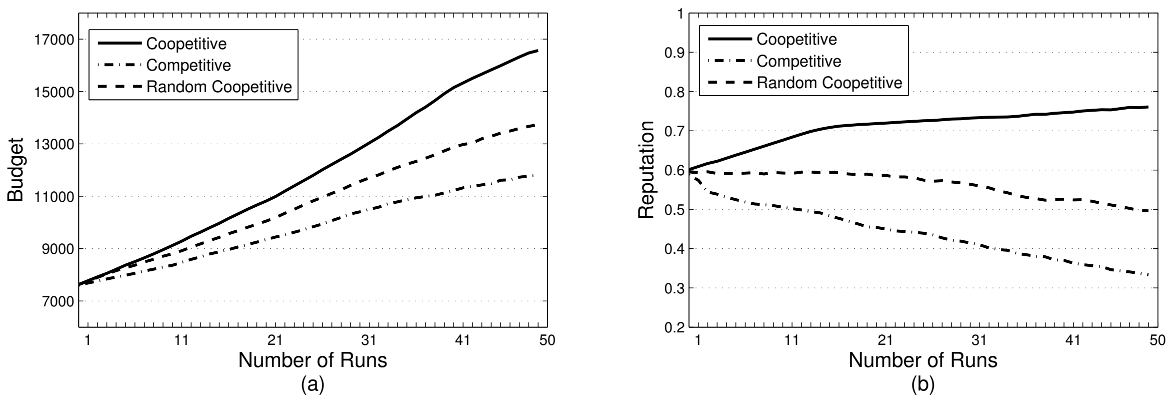


Figure 5.3: Part (a): Cumulative community budget comparison. Part (b): Average community reputation comparison over different strategic decisions.

illustrates three graphs for three different communities. Each community hosts services that follow different reasoning techniques: (1) a community that follows the interactive reasoning techniques presented in this report (referred to as coopetitive); (2) a community that follows a stochastic reasoning technique so decisions about selecting competitive or cooperative strategies are totally random (referred to as random coopetitive); and (3) a competitive community where all services follow the competitive strategy (referred to as competitive).

The results illustrated in Figure 5.3 part (a) verify the importance of the strategic decision making procedure to logically decide over the possible competitive and cooperative choices. Figure 5.3 part (b) illustrates communities average reputation of involved services. The graphs represent the influence of the rewards that the master agent uses to encourage highly capable services to compete for a task. As for the cumulative budget, we observe that the coopetitive community outperforms the random coopetitive and competitive communities in terms of average reputation. The proposed model's average reputation increases because services follow optimal strategies where they can perform better so obtain higher rewards. For the same reasons as for the cumulative budget, the average reputation of the random coopetitive community outperforms the one of the competitive community.

In our model, services are managed by selfish agents in the sense they try to maximize

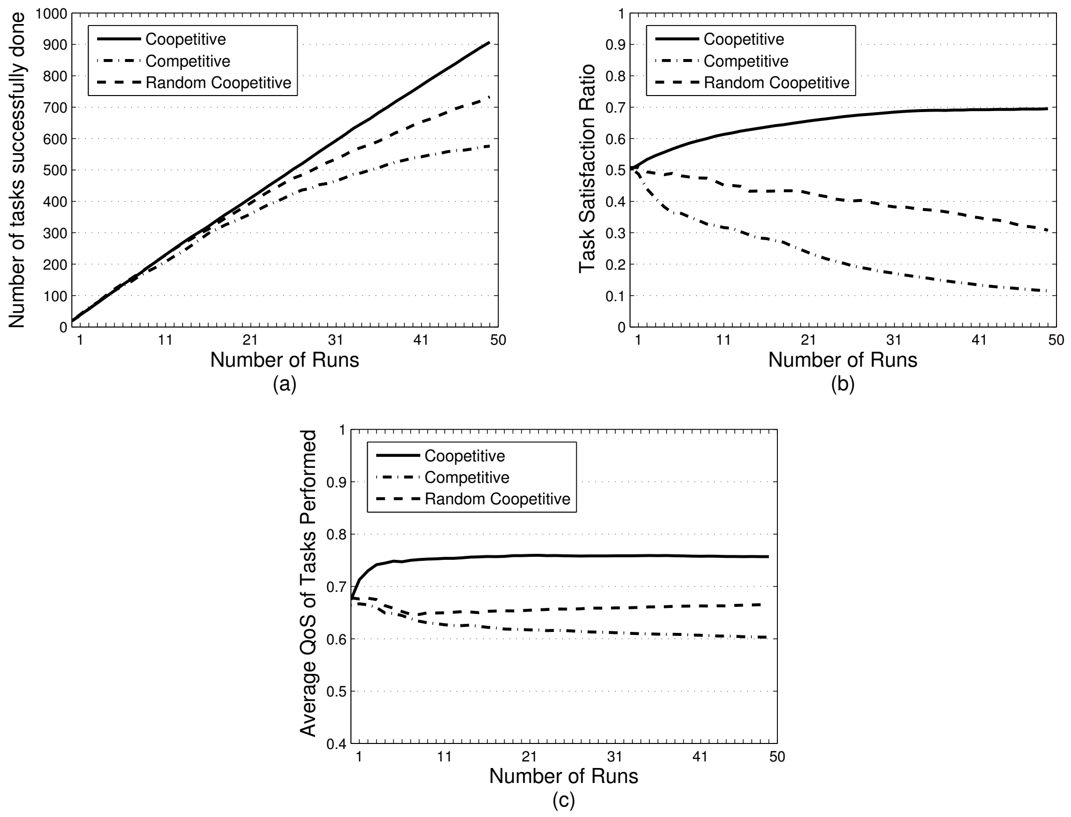


Figure 5.4: Overall performance from community's point of view. Part (a) Total number of tasks successfully done. Part (b) Ratio of tasks satisfied with required QoS. Part (c) Average QoS of performed tasks.

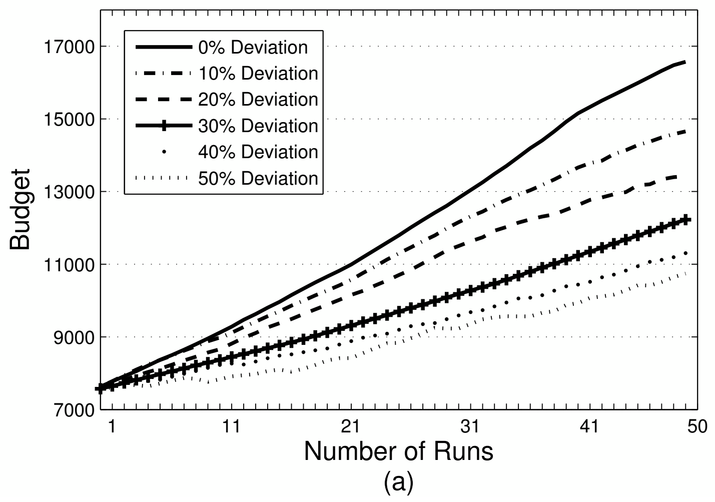
their own utilities. We analyze how their strategies affect the social welfare, and from user's and community's point of view how good the tasks are being performed. This directly impacts user's satisfaction and community's reputation in general. The Higher quality and quantity of tasks performed leads to higher user's satisfaction for the community which results in better reputation for the community. The results in Figure 5.4 show the quality and quantity of tasks being done successfully in three communities adopting the three different aforementioned strategy decision algorithms. As clearly confirmed by the simulation, the coopetitive community outperforms the stochastic and compete communities.

We conclude our analysis by discussing how effective our coopetitive decision making model is by comparing the final utility (in terms of income) of services following our model with other services deviating from that coopetitive model. In Figure 5.5 Part (a), we made the services deviate from the suggested strategy. As the simulation shows, the more services deviate from our coopetitive strategy the more they make less benefits. In Part (b), we pick one random service and simulate the scenario with its default coopetitive strategy and then we redo the simulation with exactly the same environment parameters while gradually alternating the decisions. By alternating we mean adapting the opposite of what our model does suggest. Thus, if the coopetitive model suggests to compete, the service agent will cooperate and vice versa. We run this scenario 50 times and at the end we check the service's budget and see if it gains more by deviating or not. We did this for one single deviation (i.e., alternating only one decision) to 10 different deviations (alternating 10 different decisions) during the 50 times run. As the results show, deviating from the coopetitive strategy yields less income for the service.

5.7 Summary

In this chapter, we addressed the research questions *R6* that was introduced in section 1.3. We proposed a game-theoretic based model to analyze the efficiency characteristics for the active services in open networks. The proposed framework considers the chances of web services in joining a community in different cases with truthful and lying information service agents. The proposed game analyzes the existing Nash equilibrium and situations where the maximum payoff is obtained.

Our model has the advantage of being simple and taking into account three important factors: (1) rational web services seek better status in the environment by joining the community; (2) rational web services obtain higher payoff by truth telling; and (3) the community is obtaining more effective web services. These advantages are confirmed through the conducted simulations.



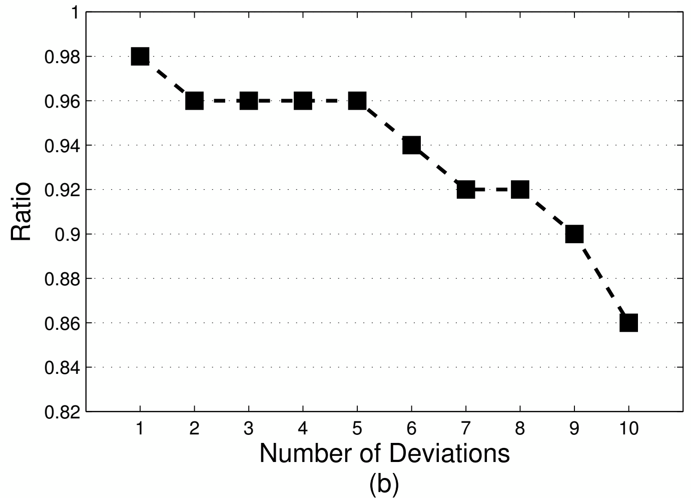


Figure 5.5: Utility loss while deviating from our coopetitive decision process. Part (a) Overall budget when deviating from our model in 0, 10, 20, 30, 40, and 50 percent of decisions. Part (b) Ratio of getting earning utility (budget) when deviating from our coopetitive strategy in 1 to 10 decisions.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we proposed three models for aggregation of web services within communities. The goal of our models are to maximize efficiency by collaborating and forming stable communities. In our first contribution, we focused on stability and fairness for all web services within the communities. In this work, we addressed the shortcomings of community formation in recent work such as considering best strategies which benefit all services involved, making solutions practical in real-time settings and taking into account the fairness and stability of communities. The proposed model offers an applicable mechanism for membership requests and selection of web services. The ultimate goal is to increase revenue by improving user satisfaction, which comes from the ability to perform more tasks with high quality. The theoretical and extensive simulation results show that our algorithms provide web services and community owners, in real-world-like environments, with applicable and near-optimal decision making mechanisms.

In our next step of research work, we proposed DDM, a strategic distributed decision making mechanism that regulates the community formation process and membership management in communities of web services. In this work, we tried to tackle the issue of autonomous web services not having a centralized architecture and complete information of all the parameters of other web services. The proposed mechanism helps web services and communities decide with whom to be grouped and cooperate. DDM first generates a trained set of data based on information obtained from large number of web services regarding their single and cooperative utilities as well as environmental parameters such as demand, service quality, etc. Communities and web services can use the trained model and instantly choose best-response strategies considering their long-term gain. In fact, the decision making mechanism is implemented as a decision tree of possible viable strategies along with their long-term expected utility. The ultimate goal of our mechanism is to make a better decision when it comes to community formation, which goes beyond short-term utility increasing choices, usually considered in the literature. We performed experiments using real date from 142 users on 4,532 web services during 64 different time slots. The experimental results show that our approach allows web services and communities, in real-world-like environments, to make near-perfect decisions. Moreover, the experiments using real data samples support the need for a long-term training model in a successful decision making process.

In our final step of the work, the focus was on inter-community interaction between services involved within the community. The contribution of this model is the proposition of a coopetitive strategic model to analyze the interacting behavior of intelligent services that are active within communities. We considered two acting strategies where service agents expect different sort of payoffs: (1) competitive strategy where the service claims that it can accomplish a task and therefore can take the responsibility over the service consumer satisfaction; and (2) cooperative strategy where the service does not take the responsibility to accomplish the task and only cooperates with competitive peers. Our proposed model advances the state-of-the-art in cooperative systems by enabling intelligent agent-based services to effectively choose their interacting strategies that lead to optimal outcomes. The proposed framework provides a reasoning technique that service agents can use to increase their overall obtained utilities. The theoretical results presented in this thesis are also backed by simulation results using a real services dataset. Moreover, we conducted extensive simulations, analyzed various scenarios, and confirmed the obtained theoretical results using parameters from a real services dataset on the web. Those results showed that our model outperforms existing competitive and random coopetitive strategies and the more services deviate from the coopetitive strategy suggested by our decision-making mechanism the less benefits they make.

This work has two main limitations. The first one is on the business side of the project. The concept of communities is still theoretical and more efforts need to be done to convince different web service providers, particularly major web service market players, to implement this concept and join different communities. However, as any new business model, once the concept of communities is commercialized and the benefits are clear for different providers, they will have enough incentive to join the communities. The other limitation is on the implementation side, more precisely the difficulty to extract real information about some quality metrics of web services and their communities. Communities of web services have not been implemented yet in large scale and real-world settings. Therefore, we had to estimate their parameters in various cases. A real-world data-set extracted from large scale online communities will help improve our results.

6.2 Future Work

As future work, for the community formation in our first model, we would like to perform more analytical and theoretical analysis on the convexity condition and also minimal

ε values in ε-*core* solution concepts based on the characteristic function in web service applications. From web service perspective, the work can be extended to consider web services compositions where a group of web services having different set of skills cooperate to perform composite tasks. Also bargaining theory from cooperative game theory concepts [82] can be used to help web services resolve the instability and unfairness issues by side payments.

For the distributed model, our future plan is to advance further the learning process on the training set we provided in our work by leveraging some game theoretical approaches. Hedonic games and fractional hedonic games [19, 10] are of particular interest where the utility of a player in a community depends on the identity of the other members of the community and the value this player ascribes to those members. We intent to investigate stability solution concepts such as Shapley value so that long-term decisions will be based on the probability that the community will last for long period. The SVM machine learning algorithm [65, 24] is suitable for classification of our training data set to better distinguish decisions based on long-term utility, as data set outputs. This can further facilitate the process of finding optimal cooperators which will result in enhancing web services' overall performance as service providers.

Publications in refereed journals and conferences

Journals

- E. Khosrowshahi Asl, J. Bentahar, H. Otrok, R. Mizouni, "Efficient Coalition Formation for Web Services", IEEE Transactions on Services Computing, 2015.

- E. Khosrowshahi Asl, J. Bentahar, H. Otrok, B. Khosravifar, R. Mizouni, "To compete or cooperate? This is the question in communities of autonomous services", Journal of Expert Systems with Applications, Elsevier, 2014.

Conferences

- E. Khosrowshahi Asl, J. Bentahar, H. Otrok, R. Mizouni, "Efficient Community Formation for Web Services", IEEE SCC, Santa Clara, CA, USA, 2013.

- B. Khoravifar, M. Alishahi, E. Khosrowshahi Asl, J. Bentahar, R. Mizouni, H. Otrok, "Analyzing Coopetition Strategies of Services within Communities", ICSOC, Shanghai, China, 2012

- O. Marey, J. Bentahar, E. Khosrowshahi Asl, M. Mbarki, R. Dssouli, "Agents' Uncertainty in Argumentation-based Negotiation: Classification and Implementation", ANT/SEIT, Hasselt, Belgium, 2014.

- H. Fallatah, J. Bentahar, E. Khosrowshahi Asl, "Social Network-Based Framework for Web Services Discovery", IEEE FiCloud, Barcelona, Spain, 2014.

Articles in process for publication in refereed journals

- E. Khosrowshahi Asl, J. Bentahar, H. Otrok, R. Mizouni, "Distributed Decision Making for Dynamic Formation of Web Services Communities", Decision Support Systems, Elsevier (Submitted: June, 2015).

Other collaborated works

- O. Marey, J. Bentahar, E. Khosrowshahi Asl, K. Soltan, R. Dssouli, "Decision making under subjective uncertainty in argumentation-based agent negotiation", Journal of Ambient Intelligence and Humanized Computing, 2015.
- F. Al-Saqqar, J. Bentahar, K. Sultan, W. Wan, E. Khosrowshahi Asl, "Model checking temporal knowledge and commitments in multi-agent systems using reduction", Journal of Simulation Modelling Practice and Theory, 2015.

Bibliography

[1] Eyhab Al-Masri and Qusay H. Mahmoud. Discovering the best web service. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 1257–1258, 2007.

[2] Eyhab Al-Masri and Qusay H. Mahmoud. Discovering the best web service: A neural network-based solution. In *SMC*, pages 4250–4255, 2009.

[3] Eduardo Adilio Pelinson Alchieri, Alysson Neves Bessani, and Joni da Silva Fraga. A dependable infrastructure for cooperative web services coordination. *Int. J. Web Service Res.*, pages 43–64, 2010.

[4] Krzysztof R. Apt and Tadeusz Radzik. Stable partitions in coalitional games. *CoRR*, abs/cs/0605132, 2006.

[5] Krzysztof R. Apt and Andreas Witzel. A generic approach to coalition formation. *IGTR*, 11(3):347–367, 2009.

[6] Danilo Ardagna and Barbara Pernici. Adaptive service composition in flexible processes. *IEEE Trans. Softw. Eng.*, 33(6):369–384, June 2007.

[7] Ehsan Khosrowshahi Asl, Jamal Bentahar, Rabeb Mizouni, Babak Khosravifar, and Hadi Otrok. To compete or cooperate? this is the question in communities of autonomous services. *Expert Syst. Appl.*, 41(10):4878–4890, 2014.

[8] Ehsan Khosrowshahi Asl, Jamal Bentahar, Rabeb Mizouni, and Hadi Otrok. Efficient community formation for web services. *Services Computing, IEEE Transactions on*, 8(4):586–600, July 2015.

[9] Ehsan Khosrowshahi Asl, Jamal Bentahar, Hadi Otrok, and Rabeb Mizouni. Efficient coalition formation for web services. In *IEEE SCC*, pages 737–744, 2013.

[10] Haris Aziz, Serge Gaspers, (Hans) Joachim Gudmundsson, Julian Mestre, and Hanjo Taubig. Welfare maximization in fractional hedonic games. In *IJCAI 2015 : International Joint Conference on Artificial Intelligence*, Buenos Aires, Argentina.

[11] Yoram Bachrach, Edith Elkind, Reshef Meir, Dmitrii Pasechnik, Michael Zuckerman, Jörg Rothe, and Jeffrey S. Rosenschein. The cost of stability in coalitional games. In *Proceedings of the 2Nd International Symposium on Algorithmic Game Theory*, SAGT '09, pages 122–134, Berlin, Heidelberg, 2009. Springer-Verlag.

[12] Matteo Baldoni, Cristina Baroglio, and Viviana Mascardi. Special issue: Agents, web services and ontologies: Integrated methodologies. *Multiagent and Grid Systems*, 6(2):103–104, 2010.

[13] Camelia Bejan and Juan Camilo Gomez. Core extensions for non-balanced tu-games. *International Journal of Game Theory*, 38(1):3–16, 2009.

[14] Boualem Benatallah, Quan Z. Sheng, and Marlon Dumas. The self-serv environment for web services composition. *IEEE Internet Computing*, 7(1):40–48, 2003.

[15] Abdelghani Benharref, Mohamed Adel Serhani, Salah Bouktif, and Jamal Bentahar. A new approach for quality enforcement in communities of web services. In Hans-Arno Jacobsen, Yang Wang, and Patrick Hung, editors, *IEEE SCC*, pages 472–479. IEEE, 2011.

[16] Karim Benouaret, Djamal Benslimane, Allel Hadjali, and Mahmoud Barhamgi. Top-k web service compositions using fuzzy dominance relationship. In *IEEE SCC*, pages 144–151, 2011.

[17] Djamal Benslimane, Zakaria Maamar, Yehia Taher, Mohammmed Lahkim, Marie Christine Fauvet, and Michael Mrissa. A Multi-Layer and Multi-Perspective Approach to Compose Web Services. In *AINA '07: Proceedings of the 21st International Conference on Advanced Networking and Applications*, pages 31–37. IEEE Computer Society, May 2007.

[18] Jamal Bentahar, Babak Khosravifar, Mohamed Adel Serhani, and Mahsa Alishahi. On the analysis of reputation for agent-based web services. *Expert Syst. Appl.*, 39(16):12438–12450, November 2012.

[19] Florian Brandl, Felix Brandt, and Martin Strobel. Fractional hedonic games: Individual and group stability. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, pages 1219–1227, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems.

[20] Ruben Casado, Muhammad Younas, and Javier Tuya. Multi-dimensional criteria for testing web services transactions. *J. Comput. Syst. Sci.*, 79(7):1057–1076, 2013.

[21] Yasmine Charif and Nicolas Sabouret. Dynamic service composition enabled by introspective agent coordination. *Autonomous Agents and Multi-Agent Systems*, 26(1):54–85, 2013.

[22] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Service Definition Language (WSDL). Technical report, March 2001.

[23] Brent N. Chun and David E. Culler. User-centric performance analysis of market-based cluster batch schedulers. In *The 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 30, 2002.

[24] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995.

[25] Davis, Michael Maschler, and M. Maschler. Existence of stable payoff configurations for cooperative. In *Essays in Mathematical Economics in Honor of Oskar Morgenstern*, pages 39–52. Princeton University Press, 1967.

[26] Haluk Demirkan and Dursun Delen. Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data in cloud. *Decision Support Systems*, 55(1):412 – 421, 2013.

[27] Xiaotie Deng and Qizhi Fang. Algorithmic cooperative game theory. *Pareto Optimality, Game Theory And Equilibria. Springer Optimization and Its Applications*, 17:159–185, 2008.

[28] Xiaotie Deng and Christos H. Papadimitriou. On the complexity of cooperative solution concepts. *Math. Oper. Res.*, 19:257–266, May 1994.

[29] Tone Dieckmann and Ulrich Schwalbe. Dynamic coalition formation and the core. *Journal of Economic Behavior and Organization*, 2002.

[30] Said Elnaffar, Zakaria Maamar, Hamdi Yahyaoui, Jamal Bentahar, and Philippe Thiran. Reputation of communities of web services - preliminary investigation. In *22nd International Conference on Advanced Information Networking and Applications, AINA 2008, Workshops Proceedings, GinoWan, Okinawa, Japan, March 25-28, 2008*, pages 1603–1608, 2008.

[31] Francois Fouss, Youssef Achbany, and Marco Saerens. A probabilistic reputation model based on transaction ratings. *Inf. Sci.*, 180(11):2095–2123, June 2010.

[32] Le Gao, Susan D. Urban, and Janani Ramachandran. A survey of transactional issues for web service composition and recovery. *Int. J. Web Grid Serv.*, 7(4):331–356, January 2011.

[33] Gianluigi Greco, Enrico Malizia, Luigi Palopoli, and Francesco Scarcello. On the complexity of the core over coalition structures. In *IJCAI*, pages 216–221, 2011.

[34] Javier Octavio Gutiérrez-García and Kwang Mong Sim. Agent-based cloud service composition. *Appl. Intell.*, 38(3):436–464, 2013.

[35] Derrick Huang and Qing Hu. Integrating web services with competitive strategies: The balanced scored approach. volume 13, pages 57–80, 2004.

[36] Audun Josang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decis. Support Syst.*, 43(2):618–644, March 2007.

[37] Radu Jurca and Boi Faltings. Reputation-based service level agreements for web services. In *Service Oriented Computing (ICSOC - 2005)*, volume 3826 of *Lecture Notes in Computer Science*, pages 396 – 409. 2005.

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

[39] Steven P. Ketchpel. The formation of coalitions among self-interested agents. In Barbara Hayes-Roth and Richard E. Korf, editors, *AAAI*, page 1467. AAAI Press / The MIT Press, 1994.

[40] Babak Khosravifar, Mahsa Alishahi, Ehsan Khosrowshahi Asl, Jamal Bentahar, Rabeb Mizouni, and Hadi Otrok. Analyzing coopetition strategies of services within communities. In *Service-Oriented Computing - 10th International Conference, ICSOC 2012, Shanghai, China, November 12-15, 2012. Proceedings*, pages 656–663, 2012.

[41] Babak Khosravifar, Mahsa Alishahi, Jamal Bentahar, and Philippe Thiran. A game theoretic approach for analyzing the efficiency of web services in collaborative networks. In *IEEE SCC*, pages 168–175, 2011.

[42] Babak Khosravifar, Jamal Bentahar, and Ahmad Moazin. Analyzing the relationships between some parameters of web services reputation. In *IEEE International Conference on Web Services, ICWS 2010, Miami, Florida, USA, July 5-10, 2010*, pages 329–336, 2010.

[43] Babak Khosravifar, Jamal Bentahar, Ahmad Moazin, Zakaria Maamar, and Philippe Thiran. Analyzing communities vs. single agent-based web services: Trust perspectives. In *IEEE SCC*, pages 194–201. IEEE Computer Society, 2010.

[44] Babak Khosravifar, Jamal Bentahar, Ahmad Moazin, and Philippe Thiran. Analyzing communities of web services using incentives. *Int. J. Web Service Res.*, 7(3):30–51, 2010.

[45] Erbin Lim, Philippe Thiran, and Zakaria Maamar. Towards defining and assessing the non-functional properties of communities of web services. In *AINA*, pages 578–585. IEEE Computer Society, 2011.

[46] Erbin Lim, Philippe Thiran, Zakaria Maamar, and Jamal Bentahar. On the analysis of satisfaction for web services selection. In *IEEE SCC*, pages 122–129, 2012.

[47] Hela Limam and Jalel Akaichi. Managing web services communities: A cache for queries optimisation. *International Journal on Web Service Computing (IJWSC)*, 1(1), 2010.

[48] An Liu, Qing Li, Liusheng Huang, Shi Ying, and Mingjun Xiao. Coalitional game for community-based autonomous web services cooperation. *IEEE Transactions on Services Computing*, 99(PrePrints), 2012.

[49] Alessio Lomuscio, Hongyang Qu, and Monika Solanki. Towards verifying compliance in agent-based web service compositions. In *AAMAS(1)*, pages 265–272, 2008.

[50] Zakaria Maamar. Web services communities : from intra-community coopetition to inter-community competition. *E-business applications for product development and competitive growth : emerging technologies.*, 2011.

[51] Zakaria Maamar, Mohammed Lahkim, Djamal Benslimane, Philippe Thiran, and Sattanathan Subramanian. Web services communities - concepts and operations. In Joaquim Filipe, Bruno Cordeiro, Jos?nd Encarnação, and Vitor Pedrosa, editors, *WEBIST (1)*, pages 323–327. INSTICC Press, 2007.

[52] Zakaria Maamar, Quan Z. Sheng, and Djamal Benslimane. Sustaining web services high-availability using communities. *2012 Seventh International Conference on Availability, Reliability and Security*, 0:834–841, 2008.

[53] Zakaria Maamar, Sattanathan Subramanian, Philippe Thiran, Djamal Benslimane, and Jamal Bentahar. An approach to engineer communities of web services: Concepts, architecture, operation, and deployment. *IJEBR*, 5(4):1–21, 2009.

[54] Zakaria Maamar, Philip Thiran, and Jamal Bentahar. Web services communities: From intra-community coopetition to inter-community competition. pages 333–343.

E-Business Application for Product Development and Competitive Growth: Emerging Technologies, 2011.

[55] Zakaria Maamar, Philippe Thiran, and Jamal Bentahar. *Web Services Communities: from Intra-Community Coopetition to Inter-Community Competition*, pages 333–344. IGI Global, 2010.

[56] Zaki Malik and Athman Bouguettaya. Evaluating rater credibility for reputation assessment of web services. In *Web Information Systems Engineering - WISE 2007, 8th International Conference on Web Information Systems Engineering, Nancy, France, December 3-7, 2007, Proceedings*, pages 38–49, 2007.

[57] William A. Massey, Geraldine A. Parker, and Ward Whitt. Estimating the parameters of a nonhomogeneous poisson process with linear rate. *Telecommunication Systems*, 5(2):361–388, 1996.

[58] E. Michael Maximilien and Munindar P. Singh. Reputation and endorsement for web services. *SIGecom Exch.*, 3(1):24–31, December 2001.

[59] Brahim Medjahed. A dynamic foundational architecture for semantic web services. *Distributed And Parallel Databases, an International Journal*, 17:179–206, 2005.

[60] Reshef Meir, Jeffrey S. Rosenschein, and Enrico Malizia. Subsidies, stability, and restricted cooperation in coalitional games. In Toby Walsh, editor, *IJCAI*, pages 301–306. IJCAI/AAAI, 2011.

[61] Daniel A. Menascé. QoS issues in web services. *IEEE Internet Computing*, 6(6):72–75, November 2002.

[62] Roger B. Myerson. *Game theory: analysis of conflict*. Harvard University Press, 1991.

[63] Cagla Okutan and Nihan Kesim Cicekli. A monolithic approach to automated composition of semantic web services with the event calculus. *Know.-Based Syst.*, 23(5):440–454, July 2010.

[64] Martin J. Osborne and Ariel Rubinstein. *A course in game theory*. The MIT Press, Cambridge, USA, 1994. electronic edition.

[65] Edgar Osuna, Robert Freund, and Federico Girosi. Training support vector machines: an application to face detection. pages 130–136, 1997.

[66] Hye-Young Paik, Boualem Benatallah, and Farouk Toumani. Toward self-organizing service communities. *Trans. Sys. Man Cyber. Part A*, 35(3):408–419, May 2005.

[67] Talal Rahwan, Tomasz P. Michalak, and Nicholas R. Jennings. Minimum search to establish worst-case guarantees in coalition structure generation. In *IJCAI*, pages 338–343, 2011.

[68] Debraj Ray. *A Game-Theoretic Perspective on Coalition Formation*. OUP Oxford, 2007.

[69] Pablo Rodriguez-Mier. Automatic web service composition with a heuristic-based search algorithm. In *IEEE ICWS*, pages 81–88, 2011.

[70] Sidney Rosario, Albert Benveniste, Stefan Haar, and Claude Jard. Probabilistic QoS and soft contracts for transaction based web services. In *2007 IEEE International Conference on Web Services (ICWS 2007), July 9-13, 2007, Salt Lake City, Utah, USA*, pages 126–133, 2007.

[71] Sidney Rosario, Albert Benveniste, Stefan Haar, and Claude Jard. Probabilistic QoS and soft contracts for transaction-based web services orchestrations. *IEEE Trans. Serv. Comput.*, 1(4):187–200, October 2008.

[72] Sidney Rosario, Albert Benveniste, and Claude Jard. Flexible probabilistic QoS management of orchestrations. *Int. J. Web Service Res.*, 7(2):21–42, 2010.

[73] Fabrizio Ruggeri and Siva Sivaganesan. On modeling change points in non-homogeneous poisson processes. *Statistical Inference for Stochastic Processes*, 8(3):311–329, 2005.

[74] Michael E. Ruth and Shengru Tu. Concurrency Issues in Automating RTS for Web Services. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 1142–1143. IEEE, July 2007.

[75] Tuomas Sandholm, Kate Larson, Martin Andersson, Onn Shehory, and Fernando Tohmé. Coalition structure generation with worst case guarantees. *Artif. Intell.*, 111(1-2):209–238, July 1999.

[76] David Schmeidler. The nucleolus of a characteristic function game. *SIAM Journal on Applied Mathematics*, pages 1163–1170, 1969.

[77] L. S. Shapley. Cores of convex games. *International Journal of Game Theory*, 1:11–26, 1971.

[78] Lloyd S. Shapley. A value for n-person games. *In Contributions to the Theory of Games (Annals of Mathematical Studies)*, 2:307–317, 1953.

[79] Lloyd S. Shapley and Martin Shubik. Quasi-cores in a monetary economy with non-convex preferences. *Econometrica*, 34(4):805–827, 1966.

[80] Xianfei Tang, Changjun Jiang, and Mengchu Zhou. Automatic web service composition based on horn clauses and petri nets. *Expert Syst. Appl.*, pages 13024–13031, 2011.

[81] Qian Tao, Hui-you Chang, Chun-qin Gu, and Yang Yi. A novel prediction approach for trustworthy QoS of web services. *Expert Syst. Appl.*, 39(3):3676–3681, February 2012.

[82] William Thomson. Bargaining and the theory of cooperative games: John Nash and beyond. RCER Working Papers 554, University of Rochester - Center for Economic Research (RCER), September 2009.

[83] Lin Kuo-Chang Wu, Shiow-yang. Structured design, consistency analysis and failure reasoning of business workflows with activity-control templates and causal ordering. *Expert Systems With Applications*, 38(7):8000–8013, 2011.

[84] Kai Xu, Qi Yu, Qing Liu, Ji Zhang, and Athman Bouguettaya. Web service management system for bioinformatics research: a case study. *Service Oriented Computing and Applications*, 5(1):1–15, 2011.

[85] Hamdi Yahyaoui. A trust-based game theoretical model for web services collaboration. *Knowl.-Based Syst.*, 27:162–169, 2012.

[86] Hamdi Yahyaoui, Zakaria Maamar, and Khouloud Boukadi. A framework to coordinate web services in composition scenarios. *IJWGS*, 6(2):95–123, 2010.

[87] Hamdi Yahyaoui and Sami Zhioua. Bootstrapping trust of web services through behavior observation. In *Web Engineering - 11th International Conference, ICWE 2011, Paphos, Cyprus, June 20-24, 2011*, pages 319–330, 2011.

[88] Makoto Yokoo, Vincent Conitzer, Tuomas Sandholm, Naoki Ohta, and Atsushi Iwasaki. Coalitional games in open anonymous environments. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI*, pages 509–515. AAAI Press / The MIT Press, 2005.

[89] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 411–421, New York, NY, USA, 2003. ACM.

[90] Yilei Zhang, Zibin Zheng, and Michael R. Lyu.

[91] Huiyuan Zheng, Weiliang Zhao, Jian Yang, and Athman Bouguettaya. QoS analysis for web service compositions with complex structures. *IEEE T. Services Computing*, 6(3):373–386, 2013.

[92] Yair Zick, Maria Polukarov, and Nicholas R. Jennings. Taxation and stability in cooperative games. In *Proc. 12th Int. Conf on Autonomous Agents and Multi-Agent Systems*, pages 523–530, 2013.