Concordia University

Electrical and Computer Engineering Department

Research Proposal

Presented in Partial Fulfillment of the Requirements

for the Degree of Doctor Philosophy in Electrical and Computer

Engineering

Cooperative Decision Making for Communities of Autonomous Web Services

by Ehsan Khosrowshahi Asl

December, 2013

This research proposal is presented before the committee composed of :

Supervisor: **Dr. Jamal Bentahar** - CIISE.

Supervisor: **Dr. Hadi Otrok** - KUSTAR(Khalifa University), ECE.

Examiners: **Dr. Peter Grogono** - CSE.

Dr. Ferhat Khendek - CSE.

Dr. Joey Paquet - CSE

Table of Contents

List of Tables

List of Figures

Abstract

Web services are loosely-coupled business applications willing to cooperate in distributed settings within different groups called communities. Communities aim to provide better visibility, efficiency, market share and total payoff. There are a number of 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.

In this proposal, we identify the problem of defining efficient algorithms for coalition formation mechanisms within communities and propose preliminary results using cooperative game-theoretic techniques. We propose a mechanism 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. The ultimate objective is to develop a mechanism for web services to form stable groups allowing them to maximize their efficiency and generate near-optimal (welfare-maximizing) communities. 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.

As remaining work in this thesis, we propose to apply different cooperative game techniques such as nucleus, kernel and bargaining solution concepts and theoretically analyze those solution concepts based on functions representing payoff and quality metrics of the communities. We also aim to apply reinforcement learning techniques for individual web service strategic decision making process.

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 cooperative game theoretic solution concepts for decision making. 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 preliminary 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 stateless 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 [30, 7]. 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 [20]. 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 [23, 5, 31]. Communities are virtual groups of web services having similar functionalities [37, 27, 24, 22], but probably different non-functional quality attributes, which form the QoS parameters. When communities are used, users send their requests to the masters of those communities, which are responsible of managing the communities, forwarding the requests to the suitable member web services and checking the credentials of those members. Communities aim to provide higher service availability and performance than what individual web services can provide. The high availability of services and the community resilience to failure are guaranteed since web services can cooperate and replace each other within the same community and since there is no single point of failure in the communities architecture.

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 service providers.

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 Motivation and Research Questions

Web service communities are dynamic by design [23]. 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 its important to consider all the parties involved in the decision making process about the community management. Most of the recent work on

communities of services are either user-centric and focus on user satisfaction [9] 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. Our initial research question is: *How can we model the community of agent-based services in order to maximize the utility of involved users, web services and community organizers?*

In order to address this problem, recently [18, 15, 19] 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 [15]. In [18], 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 and gain more, 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 just deviate and make their own group. So 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?*

In [19], 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 would like to work on is: *How can we model and analyze the cooperation among the community members in realistic, applicable and practical settings?*

1.4 Research Objectives and Contributions

In this research work, 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 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 [10], making this computation impractical in 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.

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.
- To validate the proposed methods by extensive simulations and comparison with other similar proposals.

1.5 Proposal 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. Chapter 3 provides the problem statement and presents our solution model in two different scenarios with some preliminary results. Finally, in Chapter 4, we present our conclusion, future plan, and the timetable of our research.

Chapter 2

Background and Relevant Literature

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 proposal. 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.

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 [8]. 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 proposal. 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 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 Service 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 [5, 37], 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 [21] and then comprehensively in [23] proposed an architecture utilizing *Contract-Net* protocol for engineering task distribution within communities. This architecture has been further developed in [6, 16, 17, 36]. Two types of roles have

[1] https://code.google.com/apis/console
[2] The term rational is used here in the sense that web services are utility maximizers
[3] Oxford Dictionaries

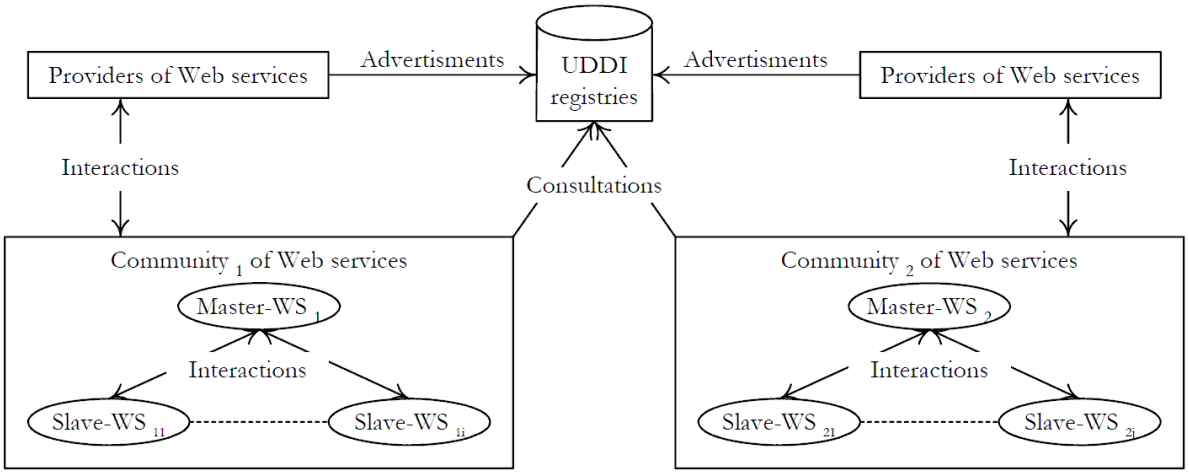

Figure 1: Communities of Web Services Architecture as Proposed in [23]

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 [23].

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

Cooperative game 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 criteria 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 [26].

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 [34]:

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

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 *Shapleyvalue* equation (1).

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

$$\forall S \subseteq N, \sum_{x_i \in S} x_i \geq v(S) \tag{2}$$

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:

$$v(S) + v(T) \leq v(S \cup T) + v(S \cap T), \forall S, T \subseteq N. \tag{3}$$

According to a classic result by Shapley [33], 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* 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).

$$\forall S \subseteq N, \sum_{x_i \in S} x_i \geq v(S) - \epsilon \tag{4}$$

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 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 non of these players can gain more than 5 by working alone. However although they have same qualities, the core can suggest a stable outcode 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 1, the two workers should each share 10 units of income, since they have the same marginal contribution to all subsets*

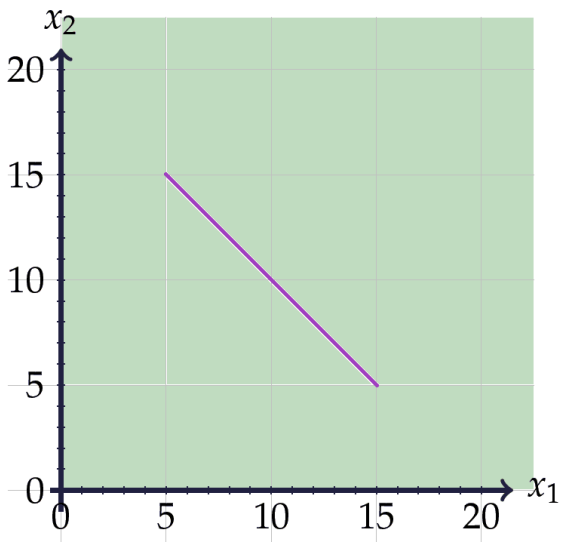

Figure 2: Core of the 2-player game of example 1

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 shapely 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, 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 Representation and Complexity Issues

Shapely 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 [35]. 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 [11]. 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 [35].

Ketchpel introduces the Bilateral Shapley Value (BSV) [14] 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

The idea of grouping web services within virtual structures was fist proposed by Zeng et al. [37]. The authors defined web service community as collection of web services providing the same functionality, but with different quality metrics. Medjahed and Boubuettaya [25] have proposed a framework and a community builder mechanism which uses semantic analysis, providing an ontological organization of web services having the same domain of interest. The community builder would suggest web services having similar operations to join the same community, or form their own community in case the semantic analysis cannot find any community that matches the service types.

Most of the recent work on communities of services are either user-centric and focus on user satisfaction [9] 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 [18, 15, 19] 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 [15]. 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 set 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 are simply ignored. The consideration of those inputs is a significant issue as existing web services can lose utility or payoff because of the new member, which can results 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 this research work.

In [18], 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 [19], 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 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 destribution method, will initially invoke the primary web service and only if the primary web service is unavailable will invoke the next backup web services as they are ordered in the coordination chain. However in cooperative models, it is preferred to have a real and active cooperative activity engaging all agents to perform the tasks more efficiently. Especially nowadays with recent advancement in cloud and hardware infrastructures availability is becoming less of an issue. So the backup web services in their model have a very low chance of getting jobs, especially

the ones further in chain, which is huge waste of web services capabilities.

2.4 Conclusive Remarks

In this research work, 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 [10], making this computation impractical in real world applications. However, using the concepts of coalition stability, we propose approximation algorithms running in polynomial time providing web services and community masters with applicable and near-optimal decision making mechanisms.

Chapter 3

Coalition Formation for Autonomous Web Services

In this chapter, we present our coalition model of agent-based web services within communities[4]. 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 some simulation results.

3.1 Preliminaries

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

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 proposal, 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.

Table 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

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 1.

We adopted a real world dataset [1] 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 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 1. We use this quality output later in evaluating the community *worth* or *payoff* function.

3.1.3 Web Service Communities

Figure 3 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

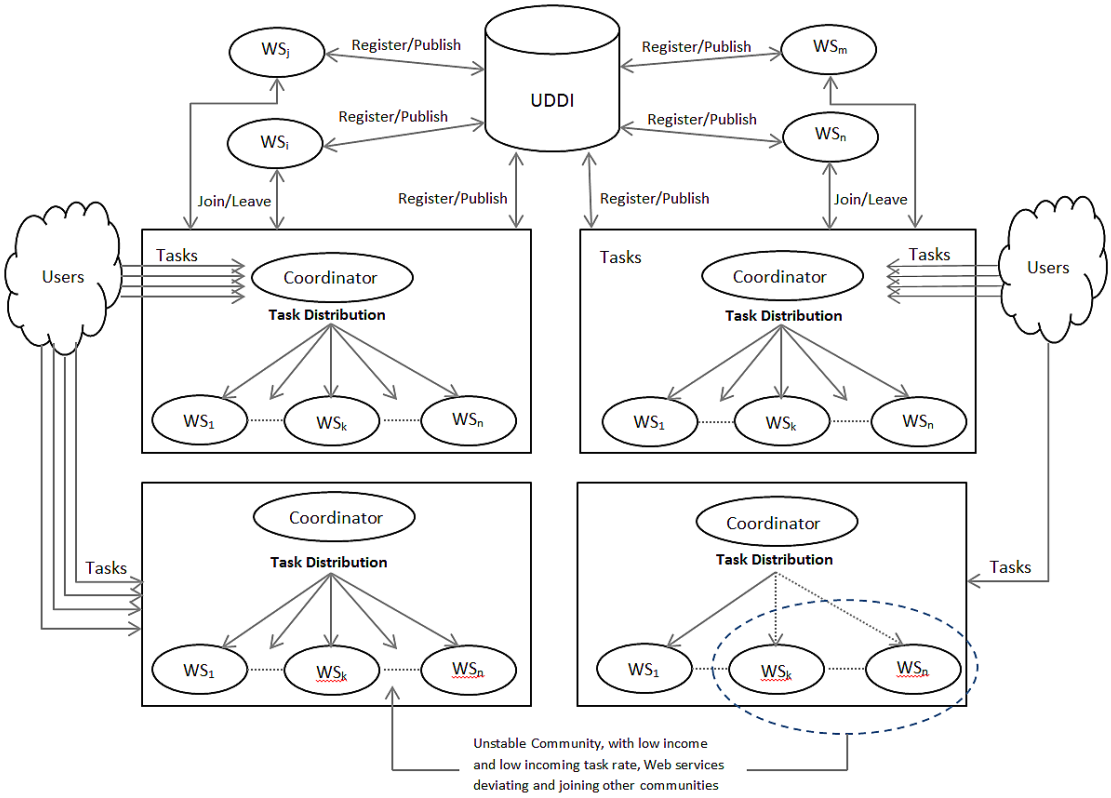

Figure 3: Architecture of Web Service communities

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.

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 service communities.

3.2.1 Task Distribution

As mentioned in Section 3.1.3, 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 but also to penalize them. 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. As mentioned in section 3.1.2, QoS_{ws} is obtained by a certain aggregation function of the parameters considered in Table 1. We have adopted a linear equal weight average over all QoS parameters listed in Table 1 excluding the *Throughput* and *Cost* parameters. A community has the option to weigh specific QoS parameters depending on 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:

Table 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

$$PO(C) = \sum_{ws \in C} (T_{ws} \times QoS_{ws}) \tag{5}$$

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.

$$Out(C) = \begin{cases} PO(C) & \text{if } \sum_{ws} Th_{ws} \leq R_C \\ PO(C) \times \frac{R_C}{\sum_{ws} Th_{ws}} & \text{if } \sum_{ws} Th_{ws} > R_C \end{cases} \tag{6}$$

The revenue function of the web service 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.

In the first example, we present the case of a community with $R_C = 10$, and three web

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: 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

services, each having different QoS_{ws} and Th_{ws} values as listed in Table 2. The worth of a community is calculated based on $Out(C)$ equation (6) 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, we even have situations where a web service (ws_3) joining a community ($\{ws_1, ws_2\}$) decreases the value of community. The reason is, the community is already

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 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

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, without hurting any other participant. In this example if we initially having two coalitions of $\{C_{master_1}, ws_2\}$

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			

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 service 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 6.

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 4).

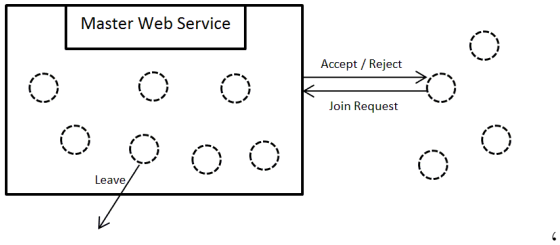

Figure 4: Web Services and A Grand Community

The membership decision is made based on throughput and QoS of the considered 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 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 [13, 26]. 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 1. *A game with a characteristic function v is convex if and only if for all S, T,*

and i *where* $S \subseteq T \subseteq N \backslash \{i\}, \forall i \in N$,

$$v(S \cup \{i\}) - v(S) \leq v(T \cup \{i\}) - v(T) \tag{7}$$

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 1 and Equation 7 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 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 community C. We call this method *Depth-2 Convex-Checker* and its run time complexity is still linear:

Theorem 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 5). 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 [32, 13, 28] 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

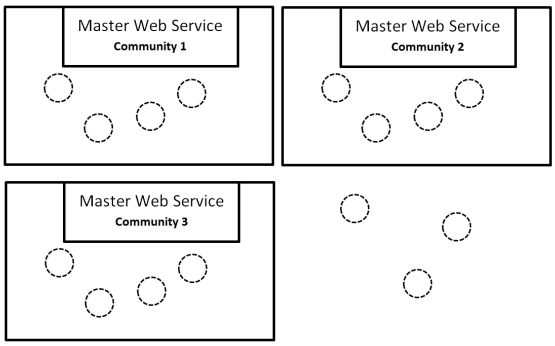

Figure 5: Web Services and Many Communities

$v_{cs}(\pi)$ 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 [3, 12, 29]. The distributed merge-and-split algorithm in [3] 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 [2], 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 [3].

3.3.3 Taxation, Subsiding and Community Stability

ssed *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 having 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. Stability is an attractive property for communities. In addition,

communities would benefit by having slightly more web services then the exact number of web services satisfying task rate cap. This is because there is always a possibility that the agents may leave the community or they may under perform and degrade the quality values they were initially performing with.

One way for communities to ensure stability could be by applying a tax, ϵ amount of cost for web services changing communities which would make deviation a costly act. However this would require all community coordinators to agree on a same amount of taxation, being governed by some external entities or web services would join communities with lesser amount of tax. Another viable solution to our scenario is to stabilize the game using external subsidies. The reason a gamWe discue is not stable is that the community is not making enough revenue to allocate enough gain to the players. A community master can subside its community with a constant coefficient value of λ.

Rewarding a community with high number of quality participants with $\lambda v(C)$, where $\lambda \leq 1$.

Obviously with a big number of λ, it is always possible to stabilize the community. However, this can be a costly act for community masters so they are interested in the minimum subside value of λ making the game stable. Subsiding or taxing in order to reduce the bargaining power of sub coalitions are called *taxation* [38] methods.

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. We

created a pool of web services and populated most of their *QoS* parameters from a real world web service dataset [1]. We implemented the simulations using Java and executed the simulations on an Intel Xeon X3450 machine with 6GBs of memory. For other parameters missing from the dataset, we used normal random distribution with parameters estimated using the method of maximum likelihood.

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 6 depicts the results of optimal ϵ-*core*, *Depth-1 Convex-Checker*, *Depth-2 Convex-Checker*, *3-Way Satisfaction* [18], and *2 Player Non-Cooperative* [15] 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, $\epsilon = 0.15 \times v(C)$, 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 community receives

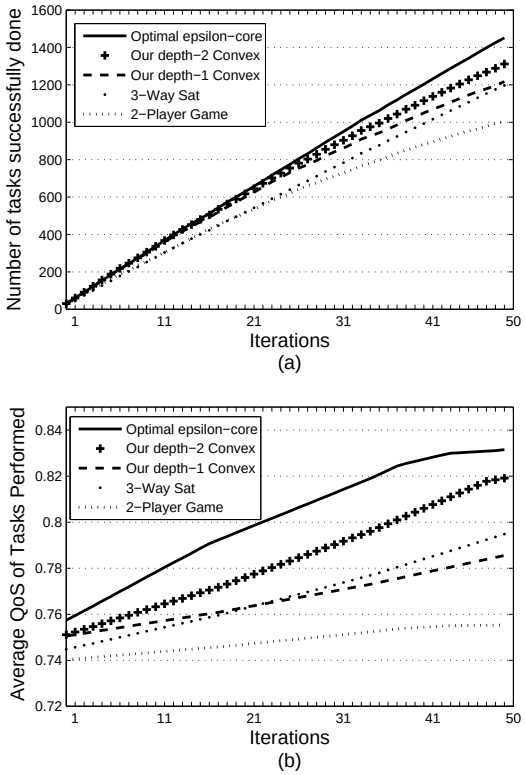

Figure 6: Part (a): Cumulative number of requests successfully done. Part (b): Average QoS of requests performed.

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 that the core is not empty. We evaluated the non-emptiness of ϵ-*core* set using the valuation

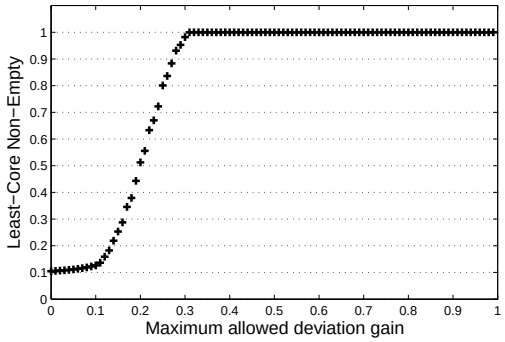

Figure 7: Analysis of ϵ-*core* set non-emptiness, for different values of ϵ

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 4), we similarly defined *relative* ϵ-*core* concept where no coalition would benefit more than $\epsilon \times v(C)$ by deviating. We set ϵ between 0 and 1 and verify the *relative* ϵ-*core* set non-emptiness. The results in Figure 7 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 Shapely 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 throughput will not contribute well to communities with low amount of user requests (low market share).

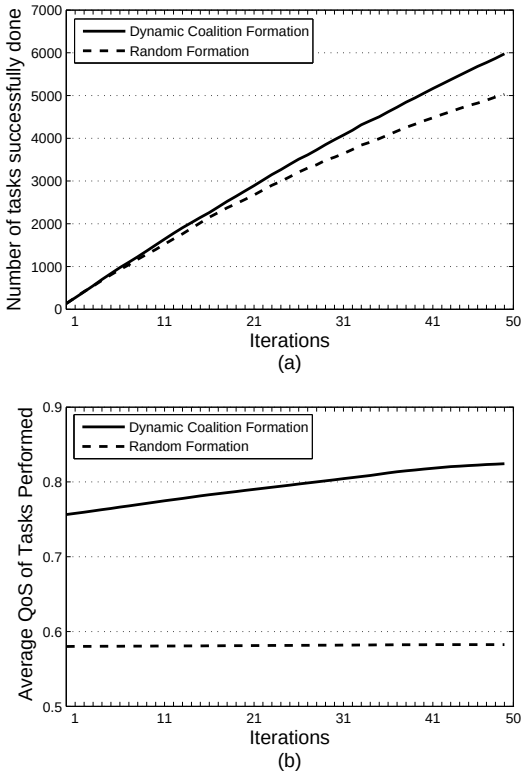

Figure 8: Part (a): Cumulative number of tasks succesfully done. Part (b): Average QoS of tasks performed.

The strong web services are likely to deviate from weak coalitions, joining a stronger one, which makes the initial coalition unstable.

In Figure 8, 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 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

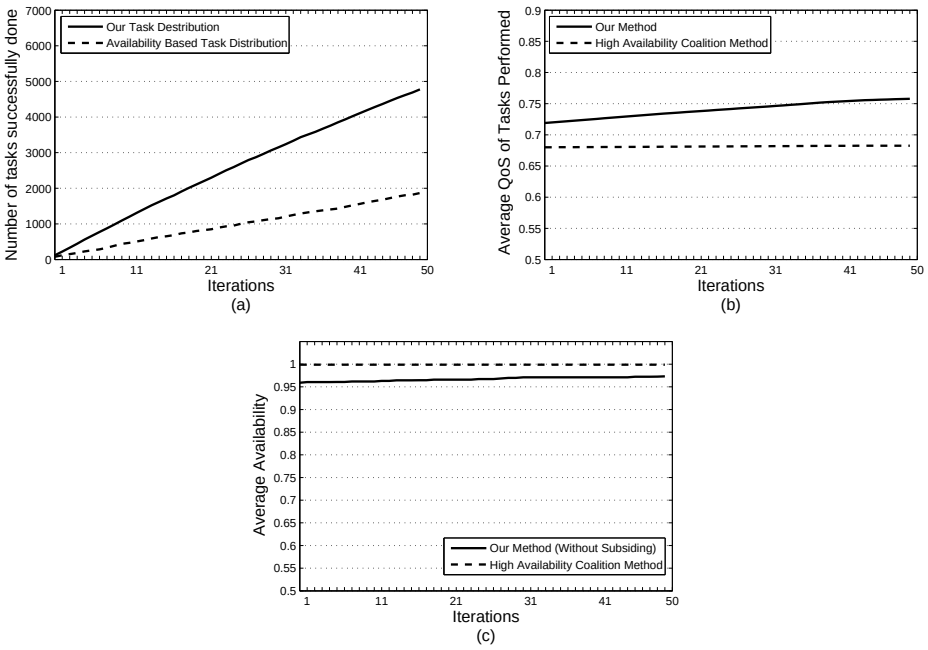

Figure 9: A comparison between our community model and the high availability community model, on communities of size 4,5,6 only. Part (a): Cumulative number of tasks successfully done. Part (b): Average QoS of tasks performed. (c) Average Community Service Availability

forms better coalitions of web services improving performance and satisfaction for both web services and coalitions.

Finally, in our last experiment, we compare our work with the work in [19] which we call it *High Availability Coalition* method, because of nature of their community valuation function, which focuses on community availability as main consideration. Their community formation model is very different than ours, however we have been very careful to make the experiment environment as fair and similar as possible. We limited our maximum community size by 5 in order to have communities with almost the same size. In their method, they have used web services as backups rather than active collaborative players, and they only get task when the first web service in an ordered chain fails to perform the task. However with recent advancement in cloud and hardware infrastructures availability

is less of an issue for web services, and web services are highly available. Part(a) of figure 9 shows our method performs almost performs tasks successfully with rate of three times more than *High Availability Coalition* method because of real cooperative nature and task distribution of our algorithm. This result shows using web services as backups, and not as real collabrative players is a huge waste of web service capability since they have very low chance of getting jobs and its the primary web service (the first in their coordination chain) which does most of the work. The average quality of service of tasks performed is also better since our method considers all quality of service metrics mentioned in Table 1. Part(c) shows the availability of community from end user point of view. The *High Availability Coalition* method almost has 100% uptime since they use web services as backups, so the chance of job failing reduces significantly as community members increase. In our method we have more chance of fail for each web service, however with some subsiding, and hiring a few more web service they can compensate the low chance of failure of web services in our community.

Chapter 4

Conclusion and Future Work

In this chapter, we describe the phases of our plan for exploring the research challenges and investigating the research issues identified in Chapter 3. This chapter also includes research methodology and future publication plan.

4.1 Conclusion

In this report, we presented research problems and questions we aim to address in the thesis and proposed some preliminary results about 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 approximation algorithms are polynomial

in complexity and provide web services and community owners with applicable and near-optimal decision making mechanisms.

4.2 Future Plan and Timeline

The future goals in this Ph.D. research work are:

- Analyzing other cooperative solution concepts such as Kernel and Nucleolus where payoff division is guaranteed to exist and may have optimal results.
- Applying other valuation function in our scenarios using for instance Weighted Voting Games (WCG) in order to develop a multiple weighted algorithm to find best coalition structures satisfying different weights on each community.
- Analyzing the impact of Q-learning and reinforcement learning on the performance of our model.
- Developing an approximation algorithm for shapely value payoff distribution vector in community setting.
- Developing a community membership algorithm technique for our agents in "incomplete information" settings.

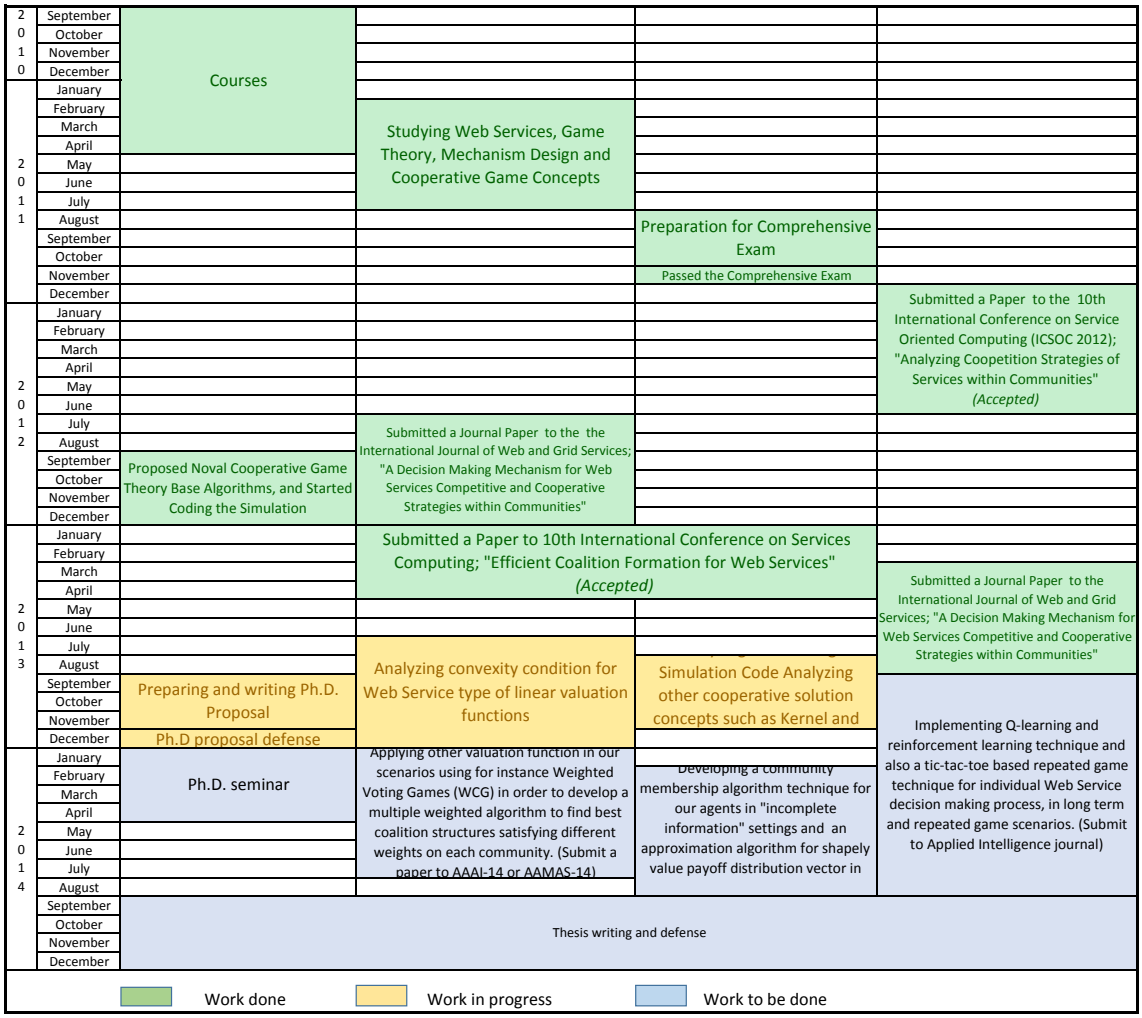

Figure 10: Research milestones and timeline

Bibliography

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

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

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

[4] E. Khosrowshahi Asl, J. Bentahar, H. Otrok, and R. Mizouni. Efficient coalition formation for web services. In *IEEE SCC*, pages 737–744, 2013.

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

[6] 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.

[7] 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.

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

[9] 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.

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

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

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

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] A. Lomuscio, H. Qu, and M. Solanki. Towards verifying compliance in agent-based web service compositions. In *AAMAS(1)*, pages 265–272, 2008.

[21] Zakaria Maamar, Mohammed Lahkim, Djamal Benslimane, Philippe Thiran, and Sattanathan Subramanian. Web services communities - concepts and operations. In Joaquim Filipe, JosÃ© Cordeiro, Bruno EncarnaÃ§Ã£o, and Vitor Pedrosa, editors, *WEBIST (1)*, pages 323–327. INSTICC Press, 2007.

[22] 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.

[23] 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.

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

[25] Brahim Medjahed and Athman Bouguettaya. A dynamic foundational architecture for semantic web services. *Distributed and Parallel Databases*, 17(2):179–206, 2005.

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

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

[28] 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.

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

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

[31] 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.

[32] 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.

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

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

[35] 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.

[36] P. Thiran Z. Maamar and J. 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.

[37] 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.

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