PhD THESIS

Semantic Service Management for Service-Oriented MAS

Author: Elena del Val Noguera

Advisors: Dr. Miguel Rebollo Pedruelo
Dr. Vicente Botti Navarro

Departamento de Sistemas Informáticos y Computación,
Universidad Politècnica de València,
Valencia, Spain

March 2013

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Resumen

Actualmente, los sistemas informáticos complejos se describen en términos de entidades que actúan como proveedores y consumidores. Estas entidades ofrecen su funcionalidad a través de servicios e interactúan entre ellas para ofrecer o pedir estos servicios. La integración de Sistemas Multi-Agente Abiertos y de Sistemas Orientados a Servicios es adecuada para implementar este tipo de sistemas. En los Sistemas Multi-Agente abiertos, los agentes entran y salen del sistema, interactúan con los demás de una manera flexible, y se consideran como entidades reactivas y proactivas, capaces de razonar acerca de lo que sucede en su entorno y llevar a cabo acciones locales sobre la base de sus observaciones para alcanzar sus metas. El área de los sistemas orientada a servicios proporciona los bloques de construcción básicos para aplicaciones empresariales complejas que son los servicios. Los servicios son independientes de la plataforma y pueden ser descubiertos y compuestos de manera dinámica. Estas características hacen que los servicios sean adecuados para hacer frente a la elevada tasa de cambios en las demandas de las empresas.

La complejidad de los sistemas informáticos, los cambios en las condiciones del entorno y el conocimiento parcial que tienen los agentes sobre el sistema requieren que los agentes cuenten con mecanismos que les faciliten tareas como el descubrimiento de servicios, la auto-organización de sus relaciones estructurales conforme se producen cambios en la demanda de servicios, y la promoción y mantenimiento de un comportamiento cooperativo entre los agentes para garantizar el buen desarrollo de la actividad del sistema.

La principal aportación de esta tesis doctoral es la propuesta de un marco para Sistemas Multi-Agente Abiertos Orientados a Servicios. Este marco integra agentes que se encuentran en una red sin ningún tipo de estructura predefinida, y agentes que

además de estar en esa red forman parte de grupos dinámicos más complejos como organizaciones virtuales. El marco que se presenta proporciona mecanismos para la gestión de las organizaciones virtuales y los servicios prestados por los agentes que forman parte de ellas. Además, el marco proporciona una estructura de red basada en homofilia entre los agentes que facilita un descubrimiento eficiente de los servicios de una manera descentralizada permitiendo a los agentes que sólo tienen una visión parcial del sistema poder localizar los recursos requeridos para alcanzar sus objetivos. Los agentes de este marco también disponen de mecanismos que les permiten razonar acerca de su ubicación en la red y de las posibles acciones estructurales para mejorar su situación en ella y mejorar el rendimiento del sistema. Por otra parte, debido a que el descubrimiento de servicios descentralizado necesita la colaboración de los agentes, el marco proporciona mecanismos para facilitar la promoción de la cooperación entre los agentes a través de la plasticidad social y los incentivos. El marco propuesto y los mecanismos que se presentan en el se han probado en diferentes escenarios con varias configuraciones y se han comparado con otros enfoques presentes en los sistemas distribuidos.

Resum

Actualment, els sistemes informàtics complexos es descriuen en termes d'entitats que actuen com proveïdors i consumidors. Aquestes entitats ofereixen la seua funcionalitat a través de serveis i interactuen entre elles per a oferir o demanar aquests serveis. La integració de Sistemes Multi-Agent Oberts i de Sistemes Orientats a Serveis es adequada per a implementar aquest tipus de sistemes. Als Sistemes Multi-Agent oberts, els agents entren i surten del sistema, interactuen amb els altres d'una manera flexible, i es consideren com a entitats reactives i proactives, capaços de raonar sobre el que passa en el seu entorn i portar a cap accions locals sobre la base de les seues observacions per assolir les seues metes. L'àrea dels Sistemes Orientats a Serveis proporciona els blocs de construcció bàsics per aplicacions empresarials complexes. Els serveis son independents de la plataforma i poden ser descoberts i compostos de manera dinàmica. Aquestes characteristiques fan que els serveis siguen adequats per a fer front de la elevada taxa de canvis en les demandes dels consumidors.

La complexitat dels sistemes informàtics, els canvis en les condicions de l'entorn i el coneixement parcial dels agents sobre el sistema mateix requereix que el agents compten amb mecanismes que els faciliten tasques com el descobriment de serveis, l'auto-organització de les seues relacions estructurals en el cas de que es produeixen canvis en la demanda de serveis, i la promoció i manteniment del comportament cooperatiu entre els agents per a garantir el bon desenvolupament de la activitat del sistema.

La principal aportació de aquesta tesi és la proposta de un marc per a sistemes multi-agent oberts orientats a serveis. Aquest marc integra els agents que es troben en una xarxa que no te una structura predefinida, i els agents que a més a més formen part de grups dinàmics i complexos com organizations virtuals. El marc proporciona mecan-

ismes per la gestió de les organitzacions virtuals i els serveis oferits pels agents que formen part d'aquestes organitzacions. A més, el marc proporciona una estructura de la xarxa que está basada en l'homofília entre els agents i que els permet un descobriment dels serveis d'una manera eficient i descentralitzada on els agents amb una visió parcial del sistema poden localitzar els recursos requerits per assolir els seus objectius. Els agents d'aquest marc també tenen mecanismes que els permeten raonar sobre la seua ubicació a la xarxa i de les possibles accions structurals per a millorar aquesta situació i així millorar el rendiment del sistema. Per altra banda, degut a que la activitat del descobriment de serveis descentralitzats necessita la col·laboració dels agents, el marc proporciona mecanismes com la plasticitat social i els incentius per a facilitar la promoció de la cooperació entre els agents. El marc proposat i els mecanismes que es presenten ací s'han provat en diferents escenaris amb diverses configurations i s'han comparat amb altres enfocaments presents en els sistemes distribuïts.

Summary

Currently, large systems are described in terms of entities that act as providers and consumers. These entities offer their functionality through services and interact to provide or ask for these services. The integration of Open Multi-Agent Systems and Service-Oriented Computing seems to be suitable to implement these systems. In open MAS, agents enter and leave the system, interact with others in a flexible way, and are considered as reactive and proactive entities that are able to reason about what is happening in their environment and perform local actions based on their observations to achieve their goals. Service-Oriented Computing provides services that are considered the basic building blocks of complex business applications. Services are platform-independent and can be discovered and composed dynamically. These features make services suitable to cope with the high rate of changes in business demands.

The openness of large systems, the changes in environment conditions, and the partial knowledge of agents about the system require agents to have mechanisms that facilitate service discovery, self-organization of structural relations as service demand changes, and the promotion and maintenance of a cooperative behavior among agents to ensure the proper performance of the service discovery activity.

The main contribution of this PhD work is the proposal of a model for Open Service-Oriented Multi-Agent Systems. This model integrates agents that are located in a network as in a plain society, and agents that also are part of dynamic complex groups (i.e., virtual organizations). The model provides mechanisms for the management of virtual organizations and services provided inside them. In addition, the model provides a network structure based on homophily between agents that model an efficient decentralized service discovery to agents that only have a partial view of the system.

Agents in this model have mechanisms that allow them to reason about their location in the network and the possible structural actions to improve their situation and the performance of the service discovery. Moreover, since decentralized service discovery relies on the cooperation of the agents, the framework provides mechanisms to facilitate the promotion of cooperation among agents through social plasticity and incentives. The proposed model and the mechanisms have been tested considering different configurations and compared with other approaches present in distributed systems.

Contents

List of Figures

List of Tables

List of Symbols

E	Entities	36
e_i	Entity	36
A	Agents	36
a_i	Agent	36
VO	Virtual Organizations	36
VO_i	Virtual Organization	36
L	Links	36
R	Roles	37
R_i	Roles played by an agent	37
N_i	Neighbors of an agent	37
st_i	Internal state of an agent	37
ϕ_i	Semantic concept of an organizational role r_i	38
S_i	Set of services of an agent	37
s_i	A single service of an agent	37
I_i	Set of inputs of the services offered by an agent	38
O_i	Set of outputs of the services offered by an agent	38

PS	Relation between success rate and average path length 109
VHN	Mixed search process that uses a combination of degree of connection and value homophily 99
CHN	Mixed search process that is based on the degree of connection and choice homophily $CH(a_i, a_j)$ 99
$\overline{k}$	Average degree of connection of agents in the network 100
$\overline{d}$	Average diameter of the network. 100
$\overline{c}$	Average clustering coefficient 100
$\overline{p}$	Average distance between all pairs of nodes 100
$\mathcal{K}_i^N$	Local knowledge of an agent about its set of direct neighbors 122
$\mathcal{K}_i^A$	Local knowledge of an agent about its set of acquaintances .. 122
$\mathcal{K}_i^E$	Local knowledge of an agent about its local environment 122
$\mathcal{K}_i^{st}$	Status of agent a_i 123
UQ_{ij}	Number of times that a query arrived to agent a_i and was not forwarded through its neighbor a_j 122
$\overline{q_i}$	Local service demand distribution 123
Q_i	Number of total queries that an agent receives and forwards . 123
ρ_i	Correlation coefficient that establishes the relationship between the local service demand distribution $\overline{q_i}$ and an estimation of the expected service demand distribution 123
$P(Q_i)$	Significance for an agent of the number of queries received . 129
$decay(UQ_{ij})$	Function that indicates the probability of maintaining a link . 131
SH_i	Structural homophily of an agent 133
$P(Q_i, clones)$	Function for evaluating the significance of the number of queries received to consider a clonning action 135
dc_i	Degree of cooperation of an agent 164
$\mathcal{C}_i$	Cooperative the behavior of an agent 164

CHAPTER

1

Introduction

1.1 Motivation

Service-Oriented Computing (SOC) and Service-Oriented Architectures (SOA) are gaining importance in industry due to their suitability for quickly coping with new business models and requirements. In these areas, services are considered to be the basic building blocks of complex business applications. Services are platform-independent and can be described, discovered, and composed dynamically. These features make services suitable for giving support to the high number of changes in business demands. In the last few years, there has been a trend in SOC to provide higher levels of functionality in order to facilitate the emergence of new services in a flexible and dynamic way exploiting existing services and avoiding the implementation of redundant services [105]. This trend brings additional considerations to the services. In order to create more complex, flexible, and adaptive systems, services

cannot simply be passive and reactive entities. They should be considered as heterogeneous entities that are reactive and proactive and that interact with other entities in a flexible way. Consequently, services are becoming more agent-like [64, 19]. Agents are autonomous, adaptive entities that are aware of what is happening in their environment and that decide to perform local actions based on their observations. Agents are able to learn from previous experiences and update and reason about their information in order to improve their decisions and achieve their goals. The integration of these two technologies results has led to Service-Oriented Multi-Agent Systems. Service-Oriented MAS are populated by agents that provide their functionality through services. They need the cooperation and collaboration with each other to achieve individual or collective goals that cannot be achieved with their individual services [63]. These agents have intelligent and social capabilities that allow them to define complex services creating Virtual Organizations (VOs). In this context, Service-Oriented MAS should provide mechanisms that: (i) facilitate the service management (service discovery and composition); (ii) facilitate the adaptation to environmental changes or new business requirements; (iii) promote and maintain the cooperation in the system. This is not an easy task due to the large number of entities, the dynamism of the systems, and the uncertainty when only partial information is available.

In complex systems such as Service-Oriented MAS, entities need to coordinate resources and services across institutional boundaries. VOs are seen as a suitable representation of a group of heterogeneous entities that collaborate in order to deal with a goal [65, 17]. These VOs could be predefined or could emerge dynamically in order to self-adjust to obtain advantages from their present context and offer compound services. Usually, the service management in VOs relies on a central entity or set of entities that have global knowledge about the available service in the VOs. This central entity provides coordination mechanisms that consider not only the functionality of agents but also organizational information [22]. These mechanisms, based on a certain desirable criteria, would be able to find adequate services in VOs and, if necessary, to construct new services through service composition. This task is

automated with the inclusion of semantic information associated to services. Semantic information facilitates the discovery and composition of services and improves the precision and accuracy of these processes [72, 88, 130]. Nevertheless, there are scenarios where a central entity is not an appropriated solution to deal with service discovery. When the number of agents increases, scalability problems appears and a central entity becomes a bottleneck. Furthermore, global knowledge is not always available and the dynamism of these systems makes the maintenance of updated information about available services complicated.

In order to avoid the dependence of an entity or a set of predefined entities responsible for facilitating the coordination among heterogeneous entities, and the maintenance of updated information, systems should provide mechanisms where the task of service management is completely decentralized. In these proposals all the entities are equal and manage their own information about their functionality and resources. They only have a partial knowledge about the system structure and available resources. In order to locate the required resources, flooding algorithms are one of the most used strategies in unstructured networks in P2P systems [82, 11]. Nevertheless, these algorithms generate too much traffic. There are other algorithms that consider information about routes stored in local registries but they are sensitive to the increase of changes in the network structure. In the last years, the use of mathematical models of Complex Networks has been proposed to create structures with certain properties that facilitate the search of resources in distributed environments in few steps taking only local information into account [70, 4, 139].

Another aspect that should be considered in Service-Oriented MAS is that agents that are part of the system, their features, and their tasks will not remain constant. These systems are expected to perform well under many circumstances, i.e., when the number of available agents changes, or when the service demand varies with time. However, the majority of the proposals for service management and specifically for service discovery in distributed systems are only focused on the location task, and do not take into consideration the inclusion of self- organization mechanisms to adapt

their structure to changes in the environmental conditions and requirements [105]. Moreover, due to in large, open, and distributed systems there is not a global view of the system, this adaptation should be performed in a decentralized way, without the supervision of any centralized authority.

In distributed environments where entities only have a partial view of the system cooperation plays a key issue. In the case of decentralized service discovery in Service-Oriented MAS, if agents do not belong to a Virtual Organization, they only know about the services they provide and their direct neighbors. Therefore, agents need the cooperation of their neighbors in order to locate the required services. However, cooperation is not always present in open and distributed systems. Non-cooperative agents pursuing their own goals could reject forwarding queries from other agents to avoid the cost of this action; therefore, the efficiency of the decentralized service discovery could be seriously damaged. As a result of this fact, mechanisms that promote the cooperation of agents in the system as well as its maintenance should be provided.

This discussion raises several questions that this research is intended to answer:

- Q1 How distributed systems deal with the problem of resource management, self-organization, and coordination between entities?
- Q2 How a service management system for open environments where individual agents and Virtual Organizations are present should be formalized?
- Q3 How the service management should be carried out in Virtual Organizations?
- Q4 How the structure of the service management and the search strategy should be in environments where only local information is available?
- Q5 Which self-organization mechanisms are necessary to maintain or improve the system efficiency when environment conditions change?

- Q6 How should the system deal with heterogeneous agents behaviors to promote cooperation in the service discovery?

- Q7 How the performance of the service management of a service-oriented MAS can be evaluated?

This work is also motivated by the research lines of the group where the PhD candidate works.[1] Among other areas of AI, the group is interested in communication, coordination and negotiation among agents, MAS architectures, integration of services and agents, MAS development methodologies and reasoning and learning in MAS. This PhD work is being developed in the context of several research projects developed by the GTI-IA group: THOMAS TIN2006-14630-C0301, OVAMAH TIN2009-13839-C03-01, CONSOLIDER INGENIO 2010 Agreement Technologies project CSD2007-00022, the GVA project PROMETEO 2008/051, Social And Economic Computing VLC-CAMPUS, and Consensus Networks for the Non-Supervised Elaboration of Agreements among Autonomous Intelligent Entities PAID-06-11-2048. The former proposes a service-based architecture for the development of virtual agent organizations. The second is intended to develop agent organizations as adaptive systems with a dynamic behavior of each entity (agent or organization). CONSOLIDER and PROMETEO projects focus on investigating the notion of agreement and all the processes and mechanisms implicated in reaching agreements between different kind of entities, analyzing them from many perspectives. Social And Economic Computing project aims the development of methods and techniques that allow the development of computational systems with social behaviors. Finally, the project of Consensus Networks focuses on the development of a model that allows agents virtual organizations to reach agreements through consensus.

[1] http://gti-ia.upv.es

1.2 Objectives

With the aim of providing answers to the questions posed in the previous section, the objective of this PhD thesis is to propose a model to facilitate the service management in Service-Oriented Multi-Agent Systems. This model integrates strategies to facilitate the service management for virtual organizations (i.e., where there is an organizational structure and there is global knowledge available about the agents, roles, and services that are part of it) or for individual agents that only have local information. Moreover, the proposed model allows agents to self-organize their structural relations and their location in the system in order to maintain the performance of the service discovery process when service demand changes. Due to the openness of Service-Oriented MAS and the requirement of cooperation to carry out the service discovery activity, the model also includes strategies that facilitate the promotion and maintenance of cooperation.

According to its main objectives, the contributions of this work are organized on different levels:

- On the *State of the Art Revision level*, an overview and analysis of different structures and search strategies present in distributed environments for the management of resources is presented. We focus on the strengths and weakness of different system structures and search techniques. Moreover, an overview of different self-organization and cooperation works is also provided. This analysis would provide the answer to question 1.

- On the *Formal level*, a proposal of a model that deals with the integration of agents, organizations, and services, being agents complex entities that collaborate to deal with changing goals in dynamic and open environments. Question 2 is answered in this level.

- On the *Organizational level*, an approach to provide coordination mechanisms in VOs in Service-Oriented MAS is presented. The mechanism is based on

automatic service discovery and composition that considers organizational information and time constraints. Question 3 is answered in this point.

- On the *Agent level* we present different proposals:
  - A loosely-structured network and search mechanisms to facilitate decentralized service management in distributed systems. This part proposes a network structure based on agents preferences and an algorithm where agents considering local information are able to find an agent that offers the service that they require. Question 4 is answered in this point.
  - Incorporation of self-organization mechanisms to adapt the system structure when network conditions change (i.e., traffic, user's demand, or available services). This point answers the question 5.
  - Mechanisms for promoting cooperation. A proposal based on the combination of a mechanism based on structural changes and a mechanism based on incentives is presented. Question 6 is answered in this point.
- On the *Evaluation level*, each proposal of the thesis has been evaluated through an application example or a set of experiments that consider different scenarios with different configuration setups. This point answers the question 7.

1.3 Structure of the Thesis

Considering the motivation and objectives of this thesis the rest of the document is structured as follows: Chapter 2 presents a state of the art in search strategies in distributed environments. We have also reviewed works in the area of self-organization and cooperation. These related works have been reviewed and analyzed to point out their main advantages and handicaps. Chapter 3 presents a formal model for Service-Oriented MAS that describes the set of components that form part of our system proposal. In Chapter 4, an instantiation of the formal model to provide a service management in VOs is presented. In Chapter 5, an instantiation of the formal model

to deal with fully decentralized environments where agents have only a partial view of the system is described. Chapter 6 presents self-organization mechanisms that allow agents to change locally its structural links in order to maintain or improve the service discovery performance when service demand changes as well as to decide if they are going to clone, remain or leave the system. In Chapter 7, the absence of cooperation problem in open systems is presented. A combination of two mechanisms is proposed to facilitate the cooperation emergence and maintenance. Finally, in Chapter 8, conclusions and future lines of work are presented.

CHAPTER

2

State of the Art

2.1 Introduction

Nowadays, there is a trend towards the design of open systems that are populated by a large number of entities that interact with each other in order to share their resources or achieve a complex goal. The entities that are part of these systems change in order to cope with environmental changes, such as new client requirements or the emergence of new business processes [142]. Therefore, under these conditions, the management of the information about which entities or resources are available in the system at a certain moment, as well as how to locate them in an efficient way are considered to be challenges.

Moreover, the structure of open systems does not remain static. Entities that are part

of the system change their structural connections with others in order to cope with new business models or requirements. Agents should incorporate self-organization mechanisms to facilitate this adaptation [39].

In open systems where there is not a central entity that controls the interactions among members of the system, the success of the service discovery process relies on the cooperation behavior of the members involved in the process. For this reason, mechanisms that promote cooperation in distributed systems are gaining importance [60].

In this chapter, we present an overview of different approaches proposed to deal with service discovery, self-organization and cooperation emergence in distributed systems. We focus specially in approaches present in the area of service discovery since it is the core of this PhD thesis. Finally, taking into account the analyzed works, we describe a set of open issues and important aspects that should be taken into consideration in this type of systems.

2.2 Search Strategies in Distributed Environments

Throughout the last decade, the research done on search strategies in distributed environments has received important contributions from traditional research areas such as Peer-to-Peer (P2P) systems [83, 136, 117, 90]. The work on this area has produced important influences in other areas that also deal with the issue of search in distributed environments, such as Service-Oriented Environments (SOE) [7, 62], and Multi-Agent Systems (MAS) [10, 35]. These areas have adapted and extended architectures and algorithms that were initially proposed in P2P to deal with specific domain requirements. Moreover, in new distributed systems there is a growing interest in the area of Complex Networks (CN) [140]. CN present new, less rigid structures that are inspired in social, biological, or technological networks, and algorithms that facilitate an efficient navigation of these networks [70].

In this section, we present an analysis of existing works that deal with search in dis-

tributed environments, such as P2P systems, Service-Oriented Environments, Multi-Agent Systems, and Complex Networks. In these areas, the search for resources, services, or entities plays a key role for the proper performance of the systems built on them. The aim of this analysis is to compare approaches from these areas taking into account the underlying system structure: *centralized*, *decentralized*, and *decentralized* approaches. Specially, we focus on decentralized approaches that are more similar to the proposal presented in this work. In Figures 2.1, 2.2, and 2.3 a schematic overview of the analysis is shown. The table groups the proposals by areas. An approach is analyzed in each row of the table. The first column describes the structure. The second column contains the first author of the article where the proposal is described and the year of the publication ([authorYear]). Moreover, the search strategy is described briefly. The three following columns describe the criteria of the structural dimension. The rest of the columns describes the search dimension. The review presented here is a short version of a previous work. We do not review all the approaches that appear on the tables. For a complete overview of search strategies in distributed environments we refer the reader to [37].

Centralized approaches such as super-peers [55], central registries[1], or middle-agents [72] are appropriate for systems with a low number of entities. In these approaches, the search process is fast and considers all the information that is available in the system. This global knowledge provides efficiency and accuracy in the search process. However, these approaches could be a bottleneck if they have a very limited capability, if the number of entities increases, or if the number of search requests and the information to take into consideration increase. Moreover, the existence of a single entity that is responsible for the management of the information about services seriously affects the robustness of the system. In order to avoid these drawbacks, distributed approaches have been proposed.

In *distributed* approaches, the responsibility of resource management relies on a set of specific entities to provide scalability and robustness. In P2P systems, structures

[1] Evolution of UDDI, White Paper , http://www.uddi.org/pubs/the_evolution_of_uddi_20020719.pdf

based on a set of *super-peers* [28] and *Distributed Hash Tables* (DHT) [131, 118, 116, 87] have been proposed. *Super-peer* approaches have problems when several super-peers fail and other peers that are less qualified must replace them. *DHT* approaches are able to locate resources in O(log n). Nevertheless, the maintenance of the indexes when the peers join and leave the system affects the performance of the system. Updates imply the interchange of messages among peers; therefore, the system could be in an inconsistent state during a period of time due to outdated references. Furthermore, these mechanisms are not very effective in locating resources with partial information. The accuracy of the search is reduced since the search is based on numeric keys and does not consider semantic information, which allows more flexible and accurate search processes. In the area of SOE there are proposals that distribute the content of the service descriptions in several registries; however there is still a central entity that coordinates, supervises, and is responsible for the maintenance of the structure [129, 108, 23]. This implies that the search process relies on this central entity and could be a critical point of failure. Some approaches that makes use of coalitions or sets of adaptative matchmakers have been proposed in MAS to provide more scalability [100]. The main problem with these approaches is that the formation process of the optimal coalition requires coordination extra effort among the entities that participate in the coalition.

There are other works based on *decentralized* structures where all the entities are considered to be equal and there is an arbitrary topology. These structures provide more flexibility and adaptability. Entities only have a partial view of the system structure or service organization and need the collaboration of the rest of the system in order to succeed in the search process. The search approaches in decentralized systems use *blind* or *informed* algorithms for locating services or resources.

Blind algorithms do not consider any information about resource locations and use *flooding* or *random* strategies. In flooding strategies, if the entity that receives the query about a resource does not have it, the entity forwards the query to all its neighbors [145, 103, 82]. The efficiency of this strategy depends on the underlying net-

work, the number of copies of one resource, and the Time To Live (TTL) that the query has. In general, flooding algorithms overload the system with the traffic generated during the search process. Random-walks have been presented as an alternative to flooding strategies [85, 147]. A random walk strategy is based on the random selection of a subset of the neighbors of the entity to forward the message. Each message follows its own path and is called a walker. A walker can be successful or fail. If the search fails, the reason could either be that the TTL has been consumed or that the query has been satisfied. This algorithm reduces the number of messages considerably when compared to flooding algorithms [14].

In order to prevent the generation of traffic, *informed* algorithms that consider local information have been proposed. These algorithms consider the information that is stored about their direct neighbors or statistics of previous searches in local registries. An example of these algorithms is presented by Crespo et al. [32]. They present a proposal that is based on *Routing indices*. These indices allow nodes to forward queries to the neighbor that is most likely to have answers. Each node has a routing index (RI) with information about the number of documents along the path and the number of documents on each topic of interest. If a node cannot answer the query, it forwards the query to a subset of its neighbors based on its local RI rather than randomly selecting or flooding the network. The problem with this proposal it keeping the large amount of information updated. The number of messages required to propagate changes in the system could overload the system. If the update process is delayed, a node can have information about routes that are not valid. Moreover, the precision of the method depends on the number of categories that are considered in the search process. Similar to the work of Crespo et al. [32], Yang et al. [145] present the *Directed Breath First* search, which forwards the queries only to a subset of neighbors considering several heuristics that are based on information from previous searches (neighbors with the highest success in previous searches, or the neighbor that finds the shortest paths, etc.). *Adaptive Probabilistic Search* is a similar approach presented by Tsoumakos et al. [134]. It is an algorithm that is based on the combination of the *k*-random walk algorithm and probabilistic forwarding. Each peer

has a local index that keeps one entry for each neighbor. The value of each entry is a tuple that contains the identifier of a neighbor and the probability that the neighbor be selected the next time based on its success in previous searches. Analogous work is presented by Kalogeraki et al. [67]. The authors propose an *Intelligent Search Mechanism* that allows peers to identify links that are likely to have relevant information. The drawback of these algorithms is that a period of time is needed to collect the information that improves the search. Moreover, if the links between peers change frequently, the statistical information stored in the local indexes could become useless. Another drawback is that some of the heuristics that are used to guide the search process could overload some peers and leave other potential peers without traffic.

Other approaches use biologically inspired techniques to locate and organize resources. For instance, *ant algorithms* are also suitable for unstructured networks because they do not rely on global knowledge about the network. The algorithm proposed by Michlmayr et al. [91] uses ants to guide the search. Each peer in the system maintains a repository of documents. Each document has the following information associated to it: a keyword, the neighbor that provides the document, and the quantity of pheromone. There are two types of ants in the system: *forward ants* and *backward ants*. The *forward ants* navigate the network until the document is found or the TTL finishes. In each step, the forward ant decides between two strategies: *exploiting* or *exploring*. The first strategy selects the best neighbor based on the quantity of pheromone. The second strategy encourages the forward ants to discover new paths. The *backward ant* is responsible for updating the path with the pheromone. The quantity of the pheromone depends on the goodness of the path. The algorithm also considers an evaporation rule to update the pheromone based on time. The main problem is that the pheromone is based on the keywords of the documents. Therefore, if a peer is looking for a document with a keyword that does not appear, even though similar documents exist, the peer will not find it in the network.

There are other approaches where the underlying structure of the system is *loosely structured* using certain criteria. This facilitates the search process. An example of

this is presented by Zhang et al. [146]. The authors propose a completely decentralized MAS without mediators. Initially, agents are connected randomly. The authors propose a reorganization algorithm to group agents with similar services together. In order to avoid isolated clusters of agents, the algorithm establishes a percentage of similar and dissimilar agents that should be in the neighborhood of the agent. For distributed searches, the authors propose the use of two algorithms: K-Nearest Neighbors (KNN) and Gradient Search Scheme (GS). The idea of the first algorithm is to redirect the queries to the most similar k-agents. In this process, the algorithm also considers the degree of the agents. The second algorithm (GS) has a first stage where it tries to find a 'good starting agent'. An agent is considered to be a 'good starting agent' if its similarity with respect to the query is above a certain threshold. If the initial agent is a 'good starting agent', the algorithm performs like KNN. Otherwise, the agent selects the most similar neighbor to the target, and a message with the similarity information is sent to that neighbor. This process is repeated n times. The agent with the highest similarity value will be choosen to restart the search using the KNN algorithm. The main disadvantage of this approach is the high cost of communication required to organize the agents into communities.

Semantics has been included in the systems in order to guide the search process, improve the accuracy of the results, and as a criterion to establish links. Upadrashta et al. [135] present a routing protocol that uses semantics included in queries to improve the performance of Gnutella systems. The main idea is that each peer keeps a list of friends and learns about their interests to obtain more relevant sources faster and with less traffic. The list reflects similarity of interests (semantic categories) between peers. Bianchini et al. [11] present a decentralized service system. Peers are connected through semantic and logical links. Semantic links are established between peers that offer similar services. Logical links are the links of the P2P system. During the search process, if a peer does not have a service that is similar to the target, it forwards the query taking into account its semantic links. If the peer does not find any semantic similar service, it queries its neighborhood. Specifically, a random subset of its neighbors in the logical layer is selected to redirect the query. This helps

to prevent the formation of isolated clusters in the semantic layer. The drawback of this approach is that the peers are organized in clusters of similar services; therefore, a peer may not be able to find services that are semantically different to its services. In this situation, the required service cannot be found using the neighbors in the semantic level and the peer must choose a neighbor using random strategies. This reduces the system to a traditional P2P system without semantics.

Basters et al. [9] use a local training set that contains previous queries and their results and semantic information about services to determine which neighbor is the most promising to forward the query to. This selection is based on probability and uses the mixed conditional bayesian risk, which considers two parameters: the semantic gain and the communication loss (number of messages to find the required service). These two parameters are calculated taking the information of the training set into account. The main drawback of this approach is that it relies on a training set that each agent maintains individually. This training set allows agents to learn which neighbor will probably return relevant semantic web services. When the agent gets into the system, this training set is empty and the agent forwards the requests using a flooding algorithm until it has enough information. In highly dynamic environments, new agents frequently join and leave the system; therefore, they will initially use flooding algorithms that overload the system.

The analysis of the different works shows that, from the *structural* point of view, the areas of P2P, SOE, and MAS follow similar structures. In systems where the number of entities is limited, centralized approaches are responsible for resource location [55, 20, 72, 111, 5]. These approaches generate less traffic, are more efficient and the results are more accurate since all the information is considered. In order to avoid bottlenecks and to provide robustness and scalability, if the system is larger, distributed approaches such as: super-peers [78] or DHT [131, 118, 116, 92, 87] in P2P; federations of registries [129] in SOE; coalitions of agents [100, 99] or distributed middle-agents [94, 124] in MAS, have been proposed. Moreover, there are some proposals that integrate structures from different areas such as DHT and Semantic

Overlay Networks based on semantic service descriptions [86, 59, 110]. Finally, if the systems are highly dynamic, with a large number of heterogeneous entities that only have partial knowledge, the search process relies on each entity. In these approaches, there are two types of search strategies: blind or informed. Blind strategies generate more traffic since they do not rely on domain-specific information. Informed strategies use statistical information from previous searches in order to guide the search. The main problem with the informed strategies is that they need a training period in order to have enough information to guide the search [145, 9]. For this reason, there are decentralized approaches that try to facilitate the search process following certain criteria to establish links between entities. An example of this is the use of Semantic Overlay Networks [82, 11]. Moreover, Complex Network provide models where short paths can be found following greedy search strategies [128, 4, 70]. These models are considered in proposals in P2P, SOE and MAS to organize and improve the resource location in decentralized and loosely structured systems [86, 54, 93].

With regard to the *search dimension*, in general, the four environments present similar solutions for dealing with the search for resources or services. In P2P systems, the majority of the proposals are oriented to the location of resources such as files, and the search process is based on keys [131, 118, 116, 87]. SOE approaches use similar structures to deal with the service discovery. However, this area introduces an important improvement: the inclusion of semantics in the service descriptions and in the search process. Semantics enhances the discovery process by providing more flexibility and precision [92, 20, 8, 111]. Moreover, semantics has also been introduced as a criterion to establish links between different entities [11, 110, 36]. In the case of MAS, the agents that populate the system offer their capabilities through services. For this reason, some of the works presented in SOE could be directly applied to solve the problem of service discovery in MAS. Moreover, agent features such as organizational roles, trust, or argumentation and negotiation capabilities have been included to improve the selection process or guide the search process [47, 21, 54].

				Structural Dimension			Search Dimension						
Area	Structure		Search Strategies	Scalability	Structural	Robustness	Adaptability	Accuracy	Traffic	Semantic	Search cost	Knowledge
Service Oriented Environments	Decentralized		[Bianchini09] Search based on semantic similarity between service descriptions + Random.	+	-	+	+	~	~	+		Local (neighbors in different layers +semantic service descriptions +Syntactic information)
			[Ding10] Local semantic and syntactic matchmaking.	-	+	+	+	+	+	+		Local (registry with potential providers of semantic services and service consumers)
			[Basters06] Flooding + probabilistic algorithm based on previous experiences stored in a local registry that consider semantic information and communication cost.	~	-	+	+	+	~	+		Local (neighbors+number of messages and semantic service descriptions)
	Distributed	Federation	[Sivashanmugam04] Search based on semantic federation domains.	~	+	~	-	~	-	~		Global (semantic inf of registries + services)
		Grid	[Skoutas08] Search based on semantic intervals.	+	+	+	+	+	~	+		Local (semantic inf about services)
		DHT	[Pirro10, He08] Search based on semantic similarity between query and services stored in peers or , DHT Chord protocol, or the combination of both.	~	~	+	+	+	~	+		Local (Finger table + Semantic Annotation Table)
		Tree	[Cao10] Search based on semantic matching in a tree of registries.	-	+	-	+	+	~	+		Global (Semantic information about services)
		Cluster	[Perryea06] search based on semantic matching in a directed graph.	-	+	-	+	+	-	+		Global (Semantic information about services)
	Centralized		[UDDI02] Direct request to the registry. Syntactic similarity between service descriptions.	-	+	-	+	+	-	-		Global (Syntactic information about services)
			[Srinivasan04] Direct request to the registry. Semantic similarity between service descriptions.	-	+	-	+	+	-	+		Global (semantic service descriptions)
		Hypergraphs	[Brogi06] Search based on semantic relations in a hypergraph.									Global(semantic service descriptions)
			[Prabhu07] Breadth First Search algorithm forward and backward simultaneously until a collision is found.									Global(semantic service descriptions)
			[Bailey06] Search based on transversals in a hypergraph.									Global(semantic service descriptions)
			[Mokhtar06] Semantic service search reduced to a numeric comparison of intervals.									Global(semantic service descriptions)

Figure 2.1: Overview of search approaches in distributed environments. An approach is analyzed in each row of the table. The first column describes the structure. The second column contains the first author of the article where the proposal is described and the year of the publication ([authorYear]). Moreover, the search strategy is described briefly. The three following columns describe the criteria of the structural dimension. The rest of the columns describes the search dimension.

Area	Structure			Search Strategies	Structural Dimension: Scalability	Structural	Robustness	Search Dimension: Adaptability	Accuracy	Traffic	Semantic	Search cost	Knowledge
P2P Networks	Decentralized	Informed		[Crespo02] Routing indexes based on number of documents of certain category in nearby nodes.	~	+	~	+	~	~	-		Local – Global (topics of interest and documents along paths)
				[Yang02] Directed breadth first where the queries are forwarded to a subset of neighbors.	+	~	+	+	~	~	-		Statistics about previous searches
				[Tsoumakos03] Adaptive probabilistic search based on the results of previous searches.	+	-	+	~	~	~	-		Local (neighbors and probabilities)
				[Michlmayr06] Ant algorithm based on keyword similarity and quantity of pheromone.				~	~	~	-		Local (keywords, documents)
		Blind		[Zhong06] random-walks biased by content popularity of the nodes.	+	-	+	+	~	+	-		Local (queries received and attended)
				[Yang02] iterative deepening: breadth-first search of a limited depth established by a poli cy.	~	-	+	+	~	+	-		Local (Files)
				[gkantsidis05, Bisnik05] k-random walk									Local
	Centralized			[Napster06] direct search request based on keywords.	-	+	-	+	~	-	-	O(1)	Global (Files – peers)
	Distributed Hierarchical	Super-peers		[Kazaa04, Morpheus, Gnutella01] broadcast among super-peers. Search based on keywords.	-	+	~	+	~	+	-	O(N)	Global (Files – peers, and references to other super-peers)
		Structure based on DHT	mesh	Tapestry [Lua05] (mesh network) search based on the degree of match between suffix digits in different levels.	~	+	~	-	~	-	-	O(log N)	Local (Files and references to other Super-peers)
				Pastry [Rowstron01] (mesh network) algorithm based on a routing table and prefix-based comparisons between IP addresses.								O(log N)	Local (leaf set, routing table, Neighborhood set)
			circular	Chord [Stoica01] (circular node Id space) finger table alg. Based on distance between keys (node id, data key).								O(log N)	Local (Finger Table)
			D-space	CAN (multidimensional id coordinates space): greedy algorithm that forwards the query to the closest coordinates in the space.								$O(d.\ N^{1/d})$	Local (a set of neighbors in the coordinate space)
			tree	Kademlia [Maymounkov02] (binary tree): algorithm based on XOR distance between identifiers of nodes.								O(log N)+c	Local (Set of tuples of near peers)
			ring	[Yu04] (Chord+semantics): 2 levels of rings. Algorithm based on semantic indexes.	~	+	~	-	~	-	+	O(log N)	Local (Finger Table)
				Symphony [Manku03] (small-world): algorithm based on distances.	+	+	+	-	~	-	-	O(1/k logN)	Local (near peers and long distance links)

Figure 2.2: Overview of search approaches in distributed environments. An approach is analyzed in each row of the table. The first column describes the structure. The second column contains the first author of the article where the proposal is described and the year of the publication ([authorYear]). Moreover, the search strategy is described briefly. The three following columns describe the criteria of the structural dimension. The rest of the columns describes the search dimension.

Area / Dim.	Structure		Search Strategies	Structural Dimension: Scalability	Structural Dimension: Robustness	Structural Dimension: Adaptability	Search Dimension: Accuracy	Search Dimension: Traffic	Search Dimension: Semantic	Search Dimension: Search cost	Search Dimension: Knowledge
MAS	Decentralized	Informed	[Zhang04] K Nearest Neighbors + Gradient Search Scheme.	+	-	~	~	+	-		Local (Vocabulary, documents, and agents With similar docs)
			[Dimakopoulos03] flooding + random algorithms.	~	~	+	~	+	-		Local (Agents, resources)
			[Babaoglu02] ant algorithm.	+	-	+	+	~	-		Local (keywords, Urls)
		Blind	[Campo02] push & pull+flooding.	-	-	+	~	-	+	-	Local (service descriptions)
			[Ouksel04] flooding algorithm.	~	~	+	~	~	+	-	Local (Ag. resource)
			[Lopes08] priority-base flooding and iterative branching depth first search.	~	-	+	+	+	+		Local (Files)
	Distributed		[Ogston01] random search and grouping procedure.	~	~	+	~	~	~	-	Local (Tasks)
			[Jha98] search in distributed matchmakers.	~	+	~	+	~	-	-	Local (Services and other matchmakers)
			[Sidgel05] search in distributed matchmakers.	+	+	~	+	~	-	-	Local (Services and other matchmakers)
			[Bromun09] semantic search in a K-dimensional tree. The search process includes argumentation and negotiation processes.	~	+	~	~	~	+	O(log N)	Local (multiple attributes about services)
	Centralized		[VazquezSalceda10] semantic search in a single matchmaker agent.	-	+	-	~	+	-	+	Global (semantic service descrip.)
			[Argente11] semantic search in a single matchmaker agent.	-	+	-	~	+	-	+	Global (semantic service descrip.)
			[Caceres06] [Klusch06][Fernandez07] semantic search in a single matchmaker agent or in a sequential combination of matchmakers.	-	+	-	+	+	-	+	Global (semantic/syntactic service descrip.)
Complex Networks	Decentralized – Loosly Structured	Scale-Free & Small World	[Barabasi99] [Adamic01] algorithm based on hop distance between nodes and connection degree.								Local (degree, number of hops)
			[Xiao06] algorithm based on three-hop information and degree.	~	~	+	~	~			Local (degree and three-hop neighbors)
			[Thadakamalla07] algorithm based on connection degree and distance.								Local (node degree and geographical situation)
			[Dell06] algorithm based on connection degree and distance between nodes.	+	-	+	-	-	-		Local (degree and distance)
			[Watts02] algorithm based on social distance between two nodes.	+	-	+	-	~	-		Local (Vector of social dimensions)
		Small-World: Social groups	[Kleinberg06] [Kleinberg01] [Adamic05] alg. based on the distance of nodes in a hierarchy.	+	~	+	~	-	-	Poly-Logarithmic	Local (social distance) Global (Hierarchy)
		Small-World: ring	[Watts98]	+	~	+	~	-	-		Local
		Small-World: lattice	[Kleinberg00] greedy algorithm based on Manhattan distance.	+	~	+	~	-	-	$\log(n)^2$	Local (Manhattan distance (links) Between neighbors)
			[Simsek.J05] search algorithm based on connection degree and similarity between nodes.	+	~	+	+	~	-		Local (similarity and degree)

Figure 2.3: Overview of search approaches in distributed environments. An approach is analyzed in each row of the table. The first column describes the structure. The second column contains the first author of the article where the proposal is described and the year of the publication ([authorYear]). Moreover, the search strategy is described briefly. The three following columns describe the criteria of the structural dimension. The rest of the columns describes the search dimension.

2.3 Self-Organization in Distributed Environments

Open systems do not remain static. They must be able to adapt themselves to changes in the environment conditions or resources demand. *Self-organization* is considered the mechanism that enables a system to arrange its organization at run-time, without explicit external commands [39]. Starting from entities that are structured in a sub-optimal organization or that are not organized at all, a self-organizing system is able to form a specific organization to pursue a well-defined goal [75]. The main issue in self-organization is to determine which is the best mechanism to reorganize the current structure through the execution of local actions to achieve the desirable behavior with a high degree of uncertainty in the system. In this context, researchers from different areas have proposed mechanisms that face the problem of self-organization. Specifically, we review some of the proposed approaches in Peer-to-Peer and Multi-Agent Systems.

Many of the proposals for self-organization in decentralized systems such as P2P are based on Ant Colony Optimization algorithms [25][48]. These algorithms are inspired on the behavior of ants and specifically on a principle called *stigmergy*. Stigmergy is an indirect mechanism of communication that is based on the information that ants leave in the environment. This information is considered by other ants in order to make decisions. Usually, works based on stigmergy propose a hybrid protocol for routing and for improving the efficiency of the paths. This protocol combines *reactive ants*, which use broadcast mechanisms for route discovery and bootstrap their routing tables, and *proactive ants*, which use unicast mechanisms based on probabilities for system maintenance.

In other approaches, peers consider trust values about their neighbors in order to decide which local actions are more appropriate to improve the structure of the system. Wang [138] presents a self-organized system of services that considers semantic information and trust in each peer of the network to form groups of peers with similar domains. The system has a hierarchical structure where 'expert-peers' contain infor-

mation about a set of peers that have information related to their domain. To build this structure and acquire knowledge about the environment, a broadcast mechanism and trust values are used by the peers. However, the hierarchical organization in peers and 'expert-peers' can overload 'expert-peers' since they initially receive and process all the queries and also the system is more sensible to deliberate attacks. Condie et. al [30] also consider trust in order to adapt a random network of peers. A peer i considers 'local trust values' with respect each peer it has interacted with. A 'local trust value' represents the number of requests that have been solved successfully by peer j (i.e., the peer that interacts with i). If a peer i has an acquaintance j that has a higher trust value that one of its current neighbors, then it changes its current link by a new one with peer j. In open environments where the peers that are part of the system changes it is difficult to establish these 'local trust values' because peers do not have information about previous direct interactions with new peers.

There are other approaches that instead of considering trust they consider similarity to decide when the local structure of peers should be re-organized. Raftopoulou and Petrakis [113] present a iCluster overlay network that manages text files. The initial structure of peers is random. The system has two global parameters that establish the number of long-links (links with dissimilar peers) and short-links (links with similar links) that a peer should have. Periodically, each peer evaluates its degree of internal clustering (degree of similarity of short-links). If the degree of internal clustering is under a threshold, the peer initiates a reorganization sending a message to m of its neighbors to find other peers enough similar to its interests and replace its current links. A drawback of this proposal is that initially nodes need to find possible candidates to create clusters through random walks which affects to the success of the searches. Another drawback is that the decision of considering re-organization of the structural links is done periodically instead of when peers consider more appropriate. Moreover, when peers cannot do a search based on similarity, they use a k-flooding algorithm that increases the traffic in the network. Another thing that this approach do not consider is the inclusion of semantic information.

Reinforcement Learning has been used in MAS to modify dynamically the behavior of the agents. In these approaches, agents adapt their behavior calculating a probability that is based on information related to its current state, previous decisions, and environment conditions [45]. Abdallah *et al.* [2] presents a self-organization mechanism where each agent, when it receives a message, updates its current state using a reinforcement algorithm and decides if it is appropriate to reorganize stochastically its current links adding or removing neighbors. The reinforcement learning algorithm used in the decision making process to update the behavior of agents is called Weighted Policy Learner (WPL). This gradient algorithm allows agents to learn stochastic policies that make agents to slow down learning when moving away from a stable policy and speedup learning when moving towards a stable policy. This approach improves previous proposals based on reinforcement learning [109] since it considers the dynamism of the network. Nevertheless, the decision making algorithm considers the reorganization of agents links based on a predefined probability. Moreover, the decision of removing neighbors is also conditioned by a constant that is dependent of the average degree of connection of the network.

There are other approaches focused in *cooperative problem solving in organizations* and how these organizations can be rearranged in order to improve its performance as the environmental conditions and the organizational goals change [75][68]. Many of these approaches rely on hierarchical structures where agents change their relations in order to distribute their workload to subordinates. The change of relations is based on a utility function that evaluates the reorganization cost, the load of the agent, and the communication cost. However, some of these models assume that all the agents are acquaintance of each other (fully connected network) that is not a realistic situation in open environments and also rely on the hierarchical structure that reduces the flexibility of the system.

In general, many of the approaches that deal with decentralized search of resources consider an initial self-organization criteria to build a suitable structure but they do not consider dynamics and how should evolve the structure as time passes and the

circumstances change. Nevertheless, there are some proposals that focus on the re-organization of links when resources change or when new peers enter or leave the system. In P2P systems, self-organization is based on the content of the peers. They consider the degree of similarity between peers or the number of successful direct interactions in order to decide if it is worthwhile to maintain a link. In the area of MAS, self-organization is not focused on the resources that agents have. Agents consider different types of links and the organizational structure. They use utility functions based on the number of times a connection has been used, the workload in certain moment, or the cost that a re-organization implies for them in order to decide if it is appropriate changing a link.

Moreover, we can conclude that there are some desirable features that distributed systems should have to provide suitable self-organization mechanisms. These main features are: (i) no external control, central authority or supervisor should guide the organization process; the organization process should be carried out locally, based on the local interactions of each agent in the system; (ii) the system should be able to evolve; and (iii) the agents of the system should be able to deal with uncertainty in order to decide when it is more appropriate to consider a self-organization decision and which action is suitable.

2.4 Cooperation Emergence

In distributed environments, cooperative behavior benefits to all individuals present in a system and implies a cost for those that have this behavior. Moreover, there is a potential exploitation of cooperative individuals by selfish ones. Selfish individuals obtain benefits from cooperator without contribute to the system. Cooperation when self-interested individuals are present in the system is known as *social dilemmas*. In these dilemmas, individuals decide between a short-term individual benefit or a long term group benefit. If there are many individuals that decide to obtain a short term benefit, this behavior can exhaust the resources and damage the performance of the

whole system.

The area of Game Theory is well-suited for understanding scenarios where the decisions and payoff or outcome of an individual depend on what other individuals are doing. Game Theory uses mathematical models that are based on well studied-games to explain these scenarios [6]. These games describe the possible available actions, the strategies, the potential payoffs of the participants, and the information considered when participants make decisions.

Furthermore, approaches based on Game Theory have been widely used to explain mechanisms through which cooperation can emerge and be maintained in different scenarios. For instance, in scenarios where individuals interact repeatedly, selfish or altruistic actions would be returned in future. In these scenarios, the mechanism to facilitate the emergence of cooperation is *direct reciprocity*. In every round, an individual has two alternatives: to cooperate or not cooperate. If the individual cooperates, the other individual may cooperate later. Hence, it might compensate to collaborate. In this scenario, the best strategy when the majority are defectors is "tit-for-tat". Otherwise, the strategy "win-stay, lose-shift" is better for maintaining cooperation [97]. When agents do not always interact with the same individuals, there are other mechanisms such as *indirect reciprocity* or *tags*. *Indirect reciprocity* is used in environments where agents interact with other agents who have information about their previous interactions with other agents. Trust and reputation are techniques that are used for indirect reciprocity [98]. Mechanisms based on *tags* facilitate the emergence of cooperation [56]. Tags are established taking into account cultural artifacts or traits [127]. *Punishment* has also been considered to promote cooperation and to overcome the "tragedy of the commons" [57]. Punishment is present in human societies where sanctioning institutions apply a punishment to those that do not obey the law. In systems where such centralized institutions do not exist, individuals are willing to punish defectors even though this implies a cost for them [58]. In general, punishment has been proven to be an efficient way to maintain cooperation [125, 126].

Many approaches that are based on games assume well-mixed populations where everybody interacts with equal frequency with everybody else. However, real populations are not well-mixed. In real populations, some individuals interact more often than others; therefore, to understand the social behavior of the systems it is important to consider the social structure. The social structure is represented by a network where links are established by the individuals following certain preferences. Therefore, the network structure influences the interactions between the individuals and the social behavior of the system. There are several works that analyze the influence of the network structure in the emergence of cooperation. These works study how structural parameters such as clustering or degree distribution affect the emergence and maintenance of cooperation [112, 101, 120, 60].

Although there are many works that take into account the structure of the networks, there are some works that not only consider the structure of the network, but also consider how local changes in the network structure can influence the collective social behavior. Eguíluz et al. [44] present a model that uses the Prisioner's Dilemma game [6] and *social plasticity* in random undirected networks of agents. Agents update their behavior in discrete time steps using an imitation strategy that considers the payoff of neighbors. The social plasticity (i.e., changes in structural links) is considered when an agent imitates a defector in order to facilitate the replacement of an unprofitable relationship with a new one that is randomly chosen. This process creates a hierarchical topology in which highly connected nodes (leaders) play an important role sustaining cooperation. Salazar et al. propose a mechanism to promote and sustain cooperation in coalitions based on taxes. Agents are located in scale-free and small-world network structures where they can change their links to join or leave a coalition [119]. Griffiths et al. [52] propose the combination of partial observation with rewiring techniques to facilitate the emergence of cooperation. They present a mechanism that is based on tags to facilitate the emergence of cooperation in a donation scenario. Their approach considers context awareness and tags of agents. Moreover, agents can remove part of their connections with agents that are not cooperative and add connections with others that can improve cooperation. There are

other approaches that also make use of rewiring techniques and partial observation to facilitate the emergence of cooperation [137]. Partial observation allows agents to observe a subset of other agents that are located outside of their circle of interaction, and afterwards, the agent imitates the majority action taken by the observed agents. Rewiring allows agents to decide to change a link after a number of unsuccessful interactions and replace it with a link to an agent recommended by a neighbor.

2.5 Final Remarks

Distributed systems are populated by a large number of heterogeneous entities that act as clients and providers. Therefore, there are a great number of different types of resources that could be considered during the search process. Moreover, the entities join and leave the system dynamically which makes the management of updated information about resources difficult. In distributed systems, there are situations where entities only have a partial view of the system, the environmental conditions change, and selfish entities appear. Taking into account these features, the systems should provide mechanisms that: (i) provide scalability and robustness when entities that participate in the search process change; (ii) locate a required resource only considering local information and do not require flooding strategies; (iii) manage different types of information (i.e., syntactic and semantic data); (iv) integrate functional and non-functional information in the selection process; (v) adapt the system structure as the environmental conditions changes (i.e., user demand, business requirements); (vi) promote the cooperation in systems where self-interest or malicious entities are present in order to maintain the system performance. In the following paragraphs we describe each feature:

Entities as agents. Entities that populate current systems could be seen as agents with complex capabilities that interact with others in order to achieve common or individual goals. Agent capabilities allow them to be aware of their situation in the

system and act in consequence. Therefore, they can incorporate some of the features that we have mentioned (i.e., trust and reputation models) in order to establish more reliable links with other entities in the system. They can also use their previous experience to improve the search process. Another interesting point to consider is the inclusion of MAS features in the search process. Most of the approaches are based on statistical or semantic information about the resources. Trust and reputation introduce new information that could give more flexibility and efficiency to the search process [54]. Moreover, negotiations [21, 81], organizational information [22], or behavioral aspects [31] could be included in the system to enhance the search process.

Robustness and Scalability. In many proposals, systems are based on rigid hierarchical structures where the content is placed on a set of entities according to hash functions. Moreover, these entities are also responsible for the search process. The most appropriate systems for providing robustness and scalability in distributed environments should be decentralized, where all the entities are equal and each one manages its own information and carries out the search process. Complex Networks provide decentralized and loosely structured models. In these models, links that follow more flexible criteria than in structured systems are established. In some approaches, semantics has been introduced to establish these links, this provides flexible self-organization and improves the query routing and search performance, facilitating the adaptation to environmental conditions as well as the search process. Moreover, there is a set of Complex Network models that have a structure where greedy algorithms can locate the target resource in short paths.

Local knowledge. In distributed systems, in most situations, entities only have a partial view of the system. Therefore, the search process should use blind strategies or informed strategies that rely on local knowledge. Blind strategies such as flooding are inefficient since they generate too much traffic. Informed algorithms are more scalable since they have information that guides the search and the number of generated messages in the process is lower than in blind strategies. For these reasons,

the use of informed strategies is more appropriate. However, not all the entities have enough information in the system to consider informed strategies. Therefore, entities should be able to choose which strategy is more appropriate by taking into account their information. In this situation, structures that consider semantic information such as Semantic Overlay Networks can guide the search when the entities do not have enough information to decide the best neighbor to forward the query to.

Data Heterogeneity. Although there are semantic-free approaches such as DHT systems that provide good performance for key discovery, they are not as efficient as semantic approaches for other types of queries such as text queries. Moreover, considering the heterogeneity of entities, an important issue is the inclusion of semantics in the search process. Semantics provides a mechanism to facilitate the interoperability of entities that require and offer services or resources [88, 142] and to improve the results of the searches. Semantics could not only be included in the service descriptions and in the search process, but also could be included in the structure of the network.Moreover, not all the semantic service descriptions are annotated using the same ontological language. Even the service descriptions that use the same language could use different ontologies that should be aligned [122]. It is nevertheless important to consider that not all the entities of the system provide semantic annotated information. For this reason, the integration of both semantic and syntactic information, as well as the use of mechanisms to align ontologies and translate different descriptions to a common model, facilitates the integration of heterogeneous entities.

Reputation. Entities cooperate with other entities in order to forward requests and provide services. In the absence of a central entity or set of entities that handles the queries and the data the entities must have mechanisms to determine which of their neighbors are trustful and whether or not to forward the query. In decentralized environments, reputation and recommendation mechanisms have been proposed to deal with this task. These mechanisms determine the trustworthiness of other entities considering direct interactions or the information received from other entities.

Reputation mechanisms could be considered during the search process as a criteria to determine which entity is more suitable to forward a query or provide a service.

Beyond functional parameters. In the majority of the proposals presented, the search process is reduced to finding an accurate result considering simple functional criteria. However, in SOE, since there is a large number of entities a set of similar service providers is easily found. Therefore, more information is necessary to determine the best provider. To carry out this ranking, non-functional parameters should be included in the service descriptions [106, 27]. However, there is no a standard way to include these parameters inside the semantic descriptions. This makes their usability in the discovery process difficult.

Self-Organization. The system structure should not be rigid: systems are not static; collaborations between entities may change; service demand can change; unexpected failures might appear; or entities might leave the system. Therefore, the structure should facilitate the adaptation at run time in order to maintain the efficiency of the search process [143]. Current proposals for self-organization focus on completely unstructured systems that use biologically inspired adaptation mechanisms or on well-defined organizational systems where adaptation protocols are defined for hierarchical structural relations. In the majority of approaches, the self-organization actions considered only focus on changes in structural relations between entities or changes in population; they do not consider the combination of the two. It would be interesting a more generic self-organization model, which combines two types of structural changes and that could be applied to systems that are loosely structured.

Cooperation. In systems where only local knowledge is available and there is no a predefined structure, the success of the search process relies on the collaboration of the entities that are part of the system. Nevertheless, this is not a very common situation in open and dynamic systems where the entities that belong to the

system frequently change. A common problem in P2P systems are the free riders. Free riders are peers that only download resources from other peers. Moreover, in MAS, self-interested agents that decide to pursue its own goals and not to collaborate to accomplish other goals are also present. Therefore, the system should provide mechanisms to deal with this problem and encourage the collaboration among agents. Malicious agents could also appear and the entities that are part of the system should be able to detect and isolate them in order to improve the efficiency of the search process. Approaches based on Game Theory have been widely used to explain mechanisms through which cooperation can emerge and be maintained in different scenarios. Mechanisms such as direct reciprocity, indirect reciprocity or tags, or punishment have been used on games that assume well-mixed populations. However, mechanisms that promote cooperation should take into account the structure of the network since it establishes the interactions between the entities and affects to the behavior of the entities of the system. In this context, there are several proposals that focus on how local changes in the network structure can influence the collective social behavior.

An approach that provides an structure and a search mechanism that includes all these features could be considered suitable to deal with the search of resources in open, dynamic and distributed environments in an efficient way.

CHAPTER

3 Abstract Model for Service-Oriented MAS

3.1 Introduction

The areas of Service Oriented Computing (SOC) and Multi-agent Systems (MAS) are getting closer. Both areas try to deal with the same kind of environments formed by loose-coupled, flexible, persistent and distributed tasks.

On the one hand, in MAS, agents must cooperate with others inside a ”society”. Due to the technological advances of recent years, the term ”society”, in which the multi-agent system participates, needs to meet several requirements such as: distribution, constant evolution, flexibility to allow members enter or exit the society, appropriate management of the organizational structure that defines the society, multi-device agent execution including devices with limited resources, and so on. All these requirements define a set of features that can be addressed through an open system paradigm and virtual organizations in multi-agent systems.

On the other hand, SOC provides the necessary support for the development of interoperable, evolvable, and massively distributed applications. Moreover, SOC brings additional considerations, such as the necessity of modeling autonomous and heterogeneous components in uncertain and dynamic environments. Such components must be autonomously reactive and proactive to interact flexibly with other components and environments. As a result, they are best thought as agents who collectively form MAS. SOC represents an emerging class of approaches with MAS-like characteristics for developing systems in large-scale open environments. Moreover, the key MAS concepts are reflected directly in SOC with ontologies, process models, choreographies, directories and facilitators, service level agreements and quality of service measures.

For these reasons it is interesting to integrate these two technologies to model service-oriented autonomous and heterogeneous computational agents in dynamic and open environments. The integration of MAS and SOC technologies creates a new type of systems called Service-Oriented Multi-Agent Systems (Service-Oriented MAS).

Service-Oriented MAS are populated by autonomous agents that offer their functionality through services. Coordination mechanisms are required in Service-Oriented MAS in order to facilitate the interaction between heterogeneous entities that take part in the system. As it has been described in Chapter 2, coordination mechanisms depend on the underlying structure and available information. There are structures where global knowledge about the entities that participate in the system and their functionality is available. In these systems, a central entity is responsible for the management of knowledge and coordination. However, there are other structures where there is only partial knowledge of the system. In these situations, coordination is more complex since only a partial view of the system is available and this introduces uncertainty in the system. In Service-Oriented MAS both situations can be found. On the one hand, there are organizations where an agent is responsible for managing the services of organizations and this agent has a global knowledge about which service is provided by each agent inside the organization. On the other

hand, there are agents that only have a local view of the capabilities of its immediate neighbors. In this case, agents should be able to locate the provider agents that offer the required services using its direct contacts. These two scenarios may be present in Service-Oriented MAS; therefore, coordination mechanisms that fulfill the needs of both should be provided. Moreover, agents should consider self-organization actions in order to rearrange its structural connections to deal with changes in the environment such as service demand. Another important aspect to consider is that cooperative behavior of the agents in Service-Oriented MAS plays an important role to ensure the correct performance of service discovery task. For this reason, the system must offer mechanisms to promote and maintain the cooperation in the system.

In this chapter we present a formal model for Service-Oriented MAS. The aim of this model is the integration of agents, services, and organizations. In this model, agents play organizational roles that define the kind of services they offer. In order to pursue a specific goal, agents can create, enter and exit from virtual organizations. Taking into account that agents can be part of a VO or can be only members of the system, the allowed actions in the system are different. We present a diagram that describes the dynamics of the system considering different states in the system.

3.2 Abstract Model for Service Management

The system is made up of a set of autonomous agents that play an organizational role in the system and offer their functionality through a set of semantic services. These agents could be members of a Virtual Organization (VO), and, therefore, they can have access to a more complete vision of the system or, at least, they have a full picture of the organization of which they are members. If agents are not part of any VO, they only have a narrow view of the global community: just a limited number of direct neighbors are known and the rest of the network remains invisible to them. In the following paragraphs the elements of the hybrid model are defined.

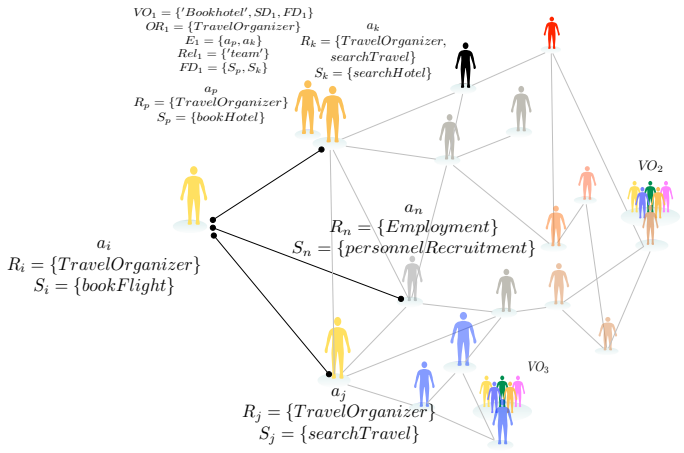


Figure 3.1: Abstract model for an open Service-Oriented MAS where nodes can be individual agents with local knowledge (i.e., direct neighbors, its roles, its services, and its internal state) or virtual organizations.

DEFINITION **1** *(System). For our proposal, the system is defined as a tuple* (E, L)*, where* $E = \{e_1, ..., e_n\} : E = A \cup VO$ *is the a finite set of entities (agents and Virtual Organizations) that are part of the system and* $L \subseteq E \times E$ *is the set of links, where each link* $(e_i, e_j) \in L$ *indicates the existence of a direct relationship between entity* e_i *and* e_j*.*

It is assumed that the knowledge relationship among entities is symmetric, so the network is an undirected graph. An entity in the system could be an agent $a_i \in A$ or a Virtual Organization $VO_i \in VO, e_i = a_i | VO_i \in E$.

In this model, an agent is a social entity that interacts with other agents in the system. It controls its own information about (i) the semantic descriptions of the services it offers, (ii) the role it plays in the organization, and (iii) knowledge about a subset of agents in the system.

DEFINITION **2** *(Agent). An agent* $a_i \in E$ *is characterized by a tuple of four elements* (R_i, S_i, N_i, st_i) *where:*

- *$R_i \subseteq R$, is the set of roles that an agent can hold at a given time, and R the roles defined in an organizational ontology $\mathcal{L}$;*
- *$S_i \subseteq S$ is the set of semantic service descriptions of the services provided by the agent and S represents all the service descriptions in the system. Each service description $s_i \in S_i$ should be associated at least to one of the roles played by the agent, $s_i \in \bigcup_{\forall r_i \in R_i} S_{r_i}, \forall s_i \in S_i$.*
- *$N_i \subseteq E$ is the set of agents an agent i can communicate with, $N_i \subseteq E - \{a_i\}$: $\forall e_j \in N_i, \exists (e_i, e_j) \in L$, and $0 < |N_i| \ll |E|$;*
- *st_i is the internal state of the agent. The internal state contains information the local environment of the agent and the degree of cooperation of the agent. See Sections 6.3 and 7.3 for a detailed view of the internal state of an agent.*

In our model, the set of neighbors N_i an agent has depends on if it is part of a virtual organization or not. If an agent is not part of any virtual organization, it only knows about the existence of a set of agents (i.e., its direct neighbors) and it is unaware of the rest of the agents in the system. If an agent is part of a virtual organization, it can have a global view of the organization (i.e., all the agents that are part of the organization) besides the agents it knew previously.

In our model, the concept of role represents the functional position of an agent. An agent must play at least one role, but it can play several roles. Moreover, a role can be played by several agents. The role determines the type of services offered by the agent. The agent acquires a role defined inside an organization of the system if it satisfies a set of requirements [46]. In our model, a role is semantically defined by an ontological concept that is defined in an organizational ontology. Moreover, a role establishes a set of semantic service descriptions that describes the functionality associated to the role.

DEFINITION **3** *(Role). A role $r_i \in R$ is defined by the tuple (ϕ_i, S_{r_i}) , where:*

- *ϕ_i is a semantic concept for the role defined using a organizational ontology $\mathcal{L}$.*
- *$S_{r_i} \in S$ is the set of semantic service descriptions associated to the role. Each service description $s_i \in S_{r_i}$ is defined by the tuple in terms of (I_i, O_i, P_i, Ef_i) (Inputs, Outputs, Preconditions and Effects).*

In our model, a Virtual Organization VO is a set of agents that need to coordinate resources and services across institutional boundaries. These agents are organized not in plain societies, but in structured organizations that enclose the real world with the society representation and ease the development of open and heterogeneous systems.

DEFINITION **4** *(Virtual Organization). A VO_i is defined as $(\mathcal{G}_i, SD_i, FD_i)$, where:*

- *$\mathcal{G}_i = \{g_i, ..., g_n\}$ is the set of goals associated to the virtual organization at a given time.*
- *$SD_i = \{E_i, OR_i, Relations\}$ is the structural dimension that defines roles and relations between them. E_i refers to the set of entities (virtual organizations, agents, or both) that are inside the organization. $OR_i \subseteq R$ refers to the roles that can be played inside the organization. $Relations$ defines the relationships between roles in the organization (i.e., organizational topology).*
- *FD_i is the functional dimension. It describes the functionality of the organization, including its services at a given time.*

$$FD_i = \bigcup_{\forall e_i \in E_i} S_i \tag{3.1}$$

Once all the components of the hybrid model are defined, the possible interactions that could occur in the system between agents and organizations are explained in detail in the following section.

3.2.1 Dynamics

Agents that are part of the proposed system can be in different states depending on their actions or the events produced by the actions of other agents in the system. In order to analyze all these possible states, we propose an state-based diagram. This diagram consists of four main states that include the main activities that an agent can carry out in the system (see Figure 3.2):

- *Non-member* state: This is the start state for agents that are not member of the system and want to get into it. 'Non-member' agents should contact with another agent that is already present in the system. All agents in the system are considered as an entry point. Agents in the system could accept or reject a request from a 'non-member' to be part of the system agent. This request contains the profile description of the agent. The profile description contains the set of organizational roles that the agent plays and the set of services the agent offers. If an agent in the system accepts the request, it is responsible for looking for a neighbor to establish a link with the new member agent. Once a new agent establishes a link with a neighbor, it goes to state *Member*.
- *Member* state: An agent in this state is member of the system, and it is connected with one or more agents but it is not member of any VO. Once an agent is part of the system it has three possible states to go depending on the events that it receives, or the actions it wants to do. The possible states are: *organizational* state, *discovery* state, and *structural* state.
- *Organizational* state: If an agent wants to create a VO, or if it wants to join an existing VO and it fulfills a set of organizational requirements, an agent arrives to *Organizational* state. The main activities in this state are related to the life-cycle of a VO. From this state an agent can go to the following states:
  - *Creation* state: An agent that is member of the system creates a VO. From this state, the agent joins the created organization and goes to the state *Organization member*.

  - *Organization member* state: An agent that is member of the system joins an existing VO if it fulfills a set of organizational requirements and becomes member of a VO. Otherwise, it is only member of the system. From this state, an agent can go to the following states:
    * *Creation* state: an agent creates a new VO;
    * *Registration* state: an agent registers its services;
    * *Discovery* state: an agent gets information about registered services in a central repository;
    * *Member* state: an agent leaves the VOs it belongs;
    * *Decentralized Discovery* state: an agent participates in a decentralized service discovery process;

- *Decentralized Discovery* state: In this state, an agent that is in the system can carry out a set of tasks related to decentralized service discovery. Basically, an agent arrives to this state when it receives a query about a service. The agent analyzes the query. If it can solve the query, it informs the agent that created the query and goes to the state *Member*. If the agent cannot solve the query, it can decide to cooperate or not in forwarding the query to one of its neighbors. If an agent does not cooperate, it goes to the state *Member*. Otherwise, it goes to the *Forwarding* state where it forwards the query to a neighbor agent that it estimates that is close to the target agent that offers the required service. Once this local decision process finishes, the agent can go to *Organization member*/*Member* state or it can analyze its structural situation in the system and goes to the *Structural* state.

- *Structural* state: In this state, an agent tries to improve the structural organization of the system through self-organization mechanisms. An agent analyzes its local information about the service demand and decides to continue, clone itself, leave the system, or change its links.

  - *Structural analysis* state: an agent arrives to update its local information and to evaluate if it has enough information to consider structural

changes. If the agent has enough information, it can make several decisions: continue in the system, clone itself if its services are demanded, leave the system if its services are not demanded (*Population analysis* state), or change its current neighbors (*Rewiring* state). After that, the agent goes to the state *Organization member*, if the agent belongs to a VO, otherwise, it goes to the state *Member*.

- *Population analysis* state: an agent arrives to this state to analyze if its services are demanded. In this state the agent creates a clone/s or leaves the system in order to adjust the population in the system to the service demand.
- *Rewiring* state: an agent arrives to this state when one of its links is not being used or when a neighbor is not cooperating forwarding its queries during a discovery process. Therefore, the agent tries to replace this link with a new one. After that, the agent goes to the state *Organization member*, if the agent belongs to a VO, otherwise, it goes to the state *Member*.

3.3 Conclusions

In this chapter, the abstract model of the system for service management in open Service-Oriented MAS has been presented. The model integrates the concepts of services, agents and organizations. The main components that take part in the system and their formal descriptions have been detailed in this chapter. Moreover, a diagram that describes the possible actions that can be carried out by agents depending on their state have been presented. The main states of this diagram are related to *organizational* aspects, *service discovery* activity, and *structural self-organization*. The organizational state deals with coordination tasks among agents situated in VOs. The decentralized service discovery state deals with the location of possible service providers considering local information. The self-organization state considers the actions that agents can take in order to deal with changes in the environment. These

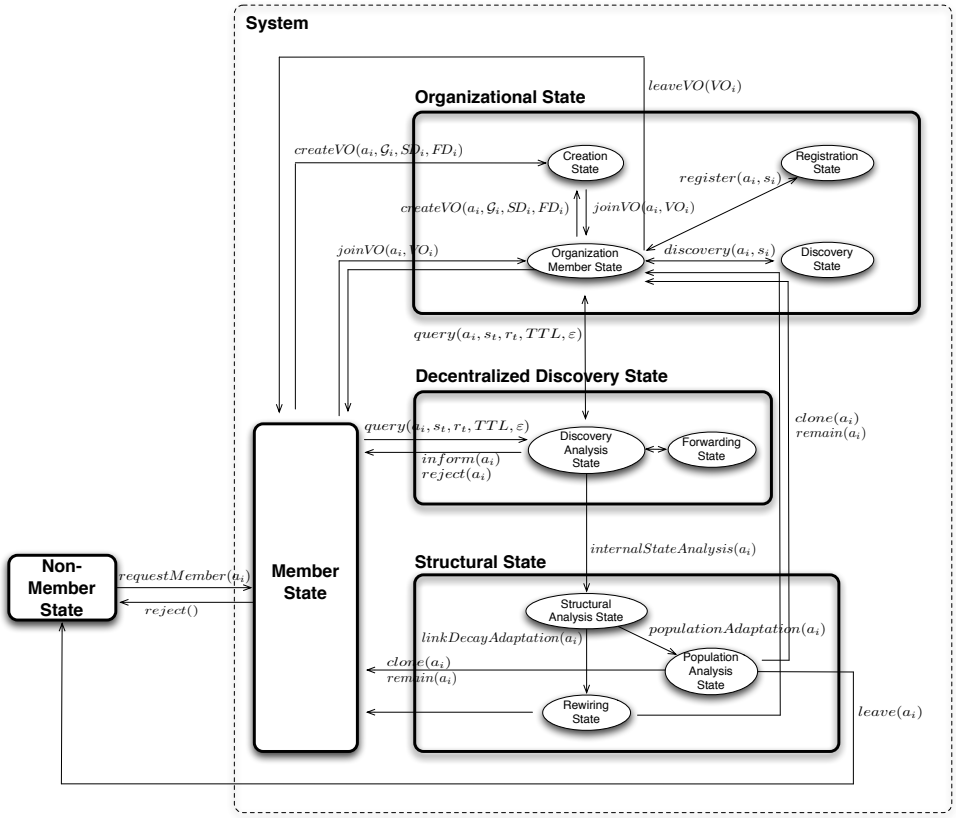


Figure 3.2: States and sub-states of agents in the system. Each state defines the allowed actions.

three states contain a set of sub-states that give more detail of the actions and events that could appear in the model.

CHAPTER

4 Service Management in Virtual Organizations

4.1 Introduction

In this chapter we present an architecture for open Service-Oriented Multi-Agent Systems. This architecture is called THOMAS (MeTHods, Techniques and Tools for Open Multi-Agent Systems). THOMAS deals with the integration of agents and services, being agents complex entities that offer their functionality through services [102]. These agents are organized not in plain societies, but in structured organizations that enclose the real world with the society representation and ease the development of open and heterogeneous systems.

Current agent platforms must integrate these concepts to allow designers employ

higher abstractions for modeling and implementing these complex systems. All these concerns are gathered in the THOMAS architecture. THOMAS architecture is integrated in the model presented in the previous chapter (see Chapter 3) in a transparent way providing the concept of Virtual Organization (VO) and facilitating the management and interaction among agents and VOs through a set of services associated to two main roles: a *service facilitator* and an *organization manager*. Specifically, we focus on the description of the services associated to the *service facilitator* role.

Moreover, in this chapter we present a framework that implements the THOMAS architecture. We implement as agents the entities responsible of the life-cycle management of virtual organizations and the management of services provided by agents and organizations. These agents play *service facilitator* and *organization manager* roles. The services they offer are implemented as semantic web services. Finally, an example of an scenario illustrates how agents are capable of discovering and employing functionalities in THOMAS framework.

This chapter is structured as follows: Section 4.2 presents the THOMAS architecture and describes the main components. Section 4.3 describes the component responsible for the service management in virtual organization. Section 4.4 presents the THOMAS framework to facilitate the creation and management of virtual organizations. Finally, Section 4.6 presents conclusions and final remarks.

4.2 Architecture Model

THOMAS architecture is based on a set of modular services that manages the life-cycle of organizations and their functionality. In THOMAS, organizations are structured by means of *virtual organizations VO*s, which represent groups of entities (agents or other units), that are related in order to pursue a common goal. Those virtual organizations have an internal topology (i.e., hierarchical, team, plain), which imposes restrictions on agent relationships and control (i.e., supervision or information relationships). An organization can also be composed of more organizations.

Moreover, *roles* are defined in each organization. They represent all required functionality needed in order to achieve the organization goal. They might also have associated *norms* for controlling role actions (i.e., which services agents playing that role are allowed to request, offer or serve; permissions for accessing resources).

Though THOMAS feeds initially on the FIPA architecture, it expands its capabilities to deal with organizations, and to boost up its services abilities. In this way, a new module in charge of managing organizations has been introduced into the architecture, along with a redefinition of the FIPA Directory Facilitator that is able to deal with services in a more elaborated way, following Service Oriented Architectures guidelines.

As it has been stated before, services are very important in this architecture. In fact, agents have access to the THOMAS infrastructure through a range of services included on different modules or components. The main components of THOMAS are the following: an agent (or set of agents) that plays the *OMS* (*'Organization Manager'*) role, an agent (or set of agents) that plays *SF* (*'Service Facilitator'*), and a *Platform Kernel* (PK) (Figure 4.1).

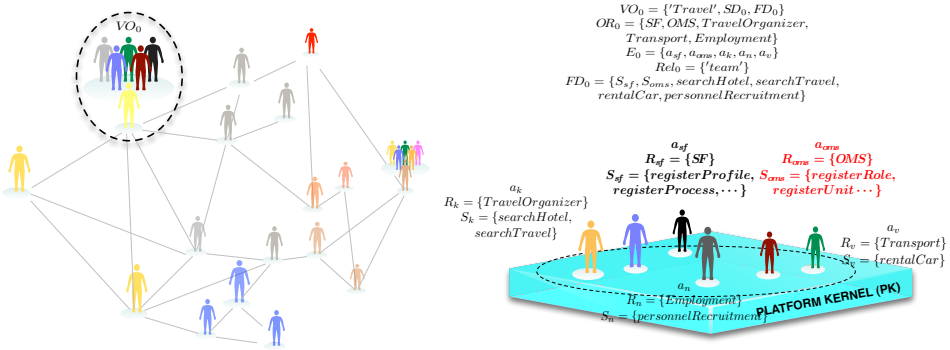


Figure 4.1: (a) Service-Oriented MAS populated by individual agents and Virtual Organizations; (b) THOMAS architecture where there are: an agent that plays the role of service facilitator (SF); an agent that plays the role of organization manager (OMS); and a platform kernel (PK).

Type	Subtype	Meta-service	Description
Structural	Registration	RegisterRole	Creates a new role inside a unit
		RegisterNorm	Includes a new norm inside a unit
		RegisterUnit	Creates a new unit inside a specific organization
		DeregisterRole	Removes a specific role description from a unit
		DeregisterNorm	Removes a specific norm description
		DeregisterUnit	Removes a unit from an organization
	Information	InformAgentRole	Indicates roles adopted by an agent
		InformMembers	Indicates entities that are members of a specific unit
		QuantityMembers	Provides the number of current members of a specific unit
		InformUnit	Provides unit description
		InformUnitRoles	Indicates which are the roles defined inside a specific unit
		InformRoleProfiles	Indicates all profiles associated to a specific role
		InformRoleNorms	Provides all norms addressed to a specific role
Dynamic	Basic	RegisterAgentRole	Creates a new <*entity, unit, role*> relationship
		DeregisterAgentRole	Removes a specific <*entity, unit, role*> relation
	Compound	AcquireRole	Requests adopting a specific role inside a unit
		LeaveRole	Requests leaving a role
		Expulse	Forces an agent to leave a specific role

Table 4.1: OMS meta-services

The *OMS* role is defined inside a VO_i and has associated services that deal with the organization management.

INSTANCE **1** *(OMS). $OMS \in R$ is a role defined inside a VO_i organization. It is*

described by the following tuple (ϕ_{oms}, S_{oms}) *where:*

- ϕ_{oms} *is the semantic concept 'Organization Manager' defined in an organizational ontology.*
- S_{oms} *contains a set of meta-services needed for organizations life-cycle management, including specification and administration of both their structural components (roles, units and norms) and their execution components (participant agents and roles they play and active organizational units). These services are classified in two groups (Table 4.1):*
  - *structural services, that modify the structural and normative organization specification;*
  - *dynamical services, that allow agents to entry or leave the organization dynamically, as well as role adoption.*

The agent that acquires the *OMS* role makes use of the following information:

- *UnitList*: it stores existing units, together with their objectives, topology and parent unit.
- *RoleList*: is stores the list of roles defined in each unit and their attributes (accessibility, visibility, position and inheritance). *Accessibility* indicates whether a role can be adopted by an agent on demand; *Visibility* indicates whether agents can obtain information of this role on demand; *Position* indicates whether it is a supervisor, subordinate or member of the unit; and *Inheritance* indicates its parent role.
- *NormList*: it stores norms defined in the system.
- *EntityPlayList*: it describes $< entity, unit, role >$ association, i.e., which roles have been adopted by an entity (agent) inside each unit.

The *SF* role is defined inside a VO_i and has associated meta-services that deal with the service management tasks.

INSTANCE **2** *(SF). $SF \in R$ is a role defined by the following tuple (ϕ_{sf}, S_{sf}) where:*

- *ϕ_{sf} is the semantic concept 'Service Facilitator' defined in an organizational ontology.*
- *S_{sf} contains a set of services needed for a suitable service management and access performance. These services are also used by the rest of THOMAS components (OMS and PK) to advertise their own services. These meta-services are classified in three groups (Table 4.2):*
  - *Registration: they allow to add, modify and remove services from the SF directory. Available services are: RegisterProfile, RegisterProcess, ModifyProfile and ModifyProcess.*
  - *Affordability: for managing the association between providers and their services. Available services are: AddProvider and RemoveProvider.*
  - *Discovery: for searching and composing services as an answer to user requirements. Available services are: SearchService, GetProfile and GetProcess.*

The *Platform Kernel (PK)* is in charge of providing the usual services required in a multi-agent platform. Therefore, it is responsible for managing the life cycle of the agents included in the different organizations, and also allows to have a communication channel (incorporating several message transport mechanisms) to facilitate the interaction among entities. Furthermore, the PK offers a safe connectivity and the necessary mechanisms that allow multi-device interconnectivity. A previous security mechanism is assumed for some of the services described below, which permits to manage who can invoke each service and over whom. For example, the supervisor of

Type	Meta-service	Description
Registration	RegisterProfile	Creates a new service description (profile)
	RegisterProcess	Creates a particular implementation (process) for a service
	ModifyProfile	Modifies an existing service profile
	ModifyProcess	Modifies an existing service process
	DeregisterProfile	Removes a service description
Affordab.	AddProvider	Adds a new provider to an existing service process
	RemoveProvider	Removes a provider from a service process
Discovery	SearchService	Searches a service (or a composition of services) that satisfies the user requirements
	GetProfile	Gets the description (profile) of an specific a service
	GetProcess	Gets the implementation (process) of an specific a service

Table 4.2: SF meta-services

an organization may have the option of creating new agents inside its organization. For this, the agent should invoke the *Register* service at platform kernel level.

The PK services needed in a THOMAS infrastructure are classified in four types: (i) *Registration*: they allow to add, modify and remove native agents from the platform; (ii) *Discovery*: services to get some information about the native agents active in the platform; (iii) *Management*: services to control the activation state of native agents in the platform; (iv) *Communication*: services to communicate agents in the platform and outside it. The complete relation of the PK services is detailed in Table 4.3.

4.3 Service Facilitator (SF)

The agent that acquires the SF role deals with the service management in virtual organizations and improves traditional Directory Facilitator in the following ways:

Type	Service	Description
Registration	Register	Registers a new agent in the platform
	Deregister	Eliminates an agent registration
	Update register	Modifies the information appearing in an agent register (except the agent name).
Discovery	Agent Search	Request information from a registered agent on the platform.
	Get Description	Obtain the platform description.
Management	Suspend	Suspend the execution of an specific agent.
	Activation	Activate the execution of an agent who currently is suspended.
Communication	Send	Send a message to any agent in the platform or outside it.

Table 4.3: PK services

- includes semantic information in the service discovery algorithm;
- considers service discovery composition when there is no a single service that can provide the service requested;
- takes into account temporal information in the service descriptions during the service composition process;
- takes into account organizational information about the roles required to ask for a certain service or to provide a service;
- manages semantic descriptions about services offered and required in the organizations.

The agent that acquires the SF role maintains an internal view of the services. This internal view is an extension of the service definition specified in Definition 3. The internal view of a service is divided in two parts: one part refers to the profile of the services that represents the abstract service specification, and the second part refers to set of service processes specifications of an existing profile. Thus, services are

organized as the following tuple:

DEFINITION **5** *(Service) A service in THOMAS is defined as:*
$< ServiceID, Providers, prof >$
$Providers ::= < ProvIDList, ServImpID, proc, ground > +$
$ProvIDList ::= ProviderID+$

where:

- *ServiceID is the identifier of the service.*
- *Providers is a set of tuples composed of a Providers identifier list (ProvIDList), the service process model specification (proc), and its particular instantiation (ground).*
- *prof is the service profile that describes the service in terms of* (I_i, O_i, P_i, Ef_i) *(Inputs, Outputs, Preconditions and Effects) and non-functional attributes, in a readable way for those agents that are searching information (or matchmaking agents which act as searching service agents). This type of representation includes a description of what the service fulfills, the constraints about its applicability, the quality of service, and the requirements that clients have to satisfy in order to use the service. This definition corresponds to the definition presented in Definition 3.*
- *ProvIDList maintains a list of service provider identifiers.*
- *ServImpID is an identifier associated to an implementation of a service profile.*
- *proc describes how a client has to use the service and specifies the semantic content for using the service. If the service is composed by more than one service, the process describes the control structure that defines the logic to execute the services.*

- *ground specifies in detail how an agent can access the service. A grounding specifies a communication protocol, the message formats, the contact port and other specific details of the service.*

Considering the internal view of the services, the set of meta-services included in the SF role are described as following:

RegisterProfile: it is used when an autonomous entity (an organization or an agent) wants to register a new service description. To do this, the profile structure has to be completed in order to provide the service description.

Service 1 RegisterProfile
Precond.: $\nexists S \in SF | s_i.prof = prof$
Input: the service goal and the service profile
Output: a service ID
Effects: $\exists S \in SF | S.ServiceID = ServiceID \wedge S.prof = prof$

RegisterProcess: it is used when an agent wants to register a particular implementation of a given service. The ID of the service and the provider entity (*EntityID*) have to be specified. There could be several providers for the same service process. In this case, the first time an implementation is going to be added, the *RegisterProcess* meta-service has to be used. If other providers offer the same process model for this service, they can be attached to it by using the *AddProvider* meta-service.

Service 2 RegisterProcess
Precond.: $\exists S \in SF | S.ServiceID = ServiceID \wedge (\nexists I \in S.Providers | I.proc = proc \wedge I.ground = ground)$
Input: service ID, its process, its grounding and provider ID
Output: a unique service ID for this process (*ServImpID*)
Effects: $\exists S \in SF | S.ServiceID = ServiceID \wedge (\exists I \in S.Providers | I.ServImpID = ServImpID \wedge ProviderID \in I.ProvIDList \wedge I.proc = proc \wedge I.ground = ground)$

ModifyProfile: it is used for modifying the description (profile) of a registered service. The client specifies the part of the service to be modified (the goal or the profile). The service ID will not change.

Service 3 ModifyProfile

Precond.: $\exists S \in SF | S.ServiceID = ServiceID$
Input: ServiceID, goal and profile
Output: —
Effects: $\exists S \in SF | S.ServiceID = ServiceID \wedge S.ServGoal = ServGoal \wedge S.prof = prof$

ModifyProcess: it is used for modifying the implementation of a registered service. The client specifies the part of the service to be modified. The service ID will not change. If more than one provider implements the service, then the implementation will not be modified.

Service 4 ModifyProcess

Precond.: $\exists S \in SF | S.ServiceID = ServiceID \wedge \exists! P \in S.Providers \wedge P.ProviderID = ProviderID$
Input: ServImpID, process and grounding
Output: —
Effects: $\exists P \in Providers | P.ServImpID = ServImpID \wedge \exists! P \in S.Providers \wedge P.ProviderID = ProviderID \wedge P.proc = proc \wedge P.ServGroun = ServGroun$

DeregisterProfile: it is used for deleting a service description.

Service 5 DeregisterProfile

Precond.: $\exists S \in SF | S.ServiceID = ServiceID$
Input: a valid service ID
Output: —
Effects: $\nexists S \in SF | S.ServiceID = ServiceID$

AddProvider: adds a new provider to an existing service implementation.

Service 6 AddProvider
Precond.: $\exists P \in Providers | P.ServiceImpID = ServImpID \wedge ProviderID \notin P.ProvIDList$
Input: IDs of the service (*ServImpID*) and the provider (*ProviderID*)
Output: —
Effects: $\exists P \in Providers | P.ServImpID = ServImpID \wedge ProviderID \in P.ProvIDList$

RemoveProvider: it deletes a provider from a service implementation. If it is the last provider, then the service implementation is automatically erased. Furthermore, if that is the unique implementation of the service, then the provider is alerted and it can deregister the service.

Service 7 RemoveProvider
Precond.: $\exists P \in Providers | P.ServImpID = ServImpID \wedge ProviderID \in P.ProvIDList$
Input: IDs of the service (*ServImpID*) and the provider (*ProviderID*)
Output: —
Effects:

1. $\exists P \in Providers | P.ServImpID = ServImpID \wedge ProviderID \notin P.ProvIDList$
2. $\exists P \in Providers | P.ProvIDList = \emptyset \rightarrow [ModifyProcess(P.ServImpID, \emptyset, \emptyset, \emptyset)]$
3. $\exists S \in SF | S.Providers = \emptyset \rightarrow [Deregister(S.ServiceID)]$

SearchService: it searches a service whose description satisfies the client request. The search process can use matchmaking, composition and other techniques to solve complex queries. To request a service, the user has to specify a *ServicePurpose*. It is a general structure in which the request is stored. It can be expressed as a *ServiceGoal*, a partial *ServiceProfile* description or a combination of both. The result of the search is a list of tuples $< ServiceID, Ranking >$, where ranking indicates the matching degree between the service and the request.

Service 8 SearchService
Precond.: —
Input: ServicePurpose
Output: list of tuples $< ServiceID, Ranking >$
Effects: —

GetProfile: it is used to retrieve the profile details (description) for an specific service.

Service 9 GetProfile
Precond.: $\exists serv \in SF | serv.ServiceID = ServiceID$
Input: a valid service ID
Output: service profile and goal
Effects: —

GetProcess: it is used to retrieve the process details (implementation) for an specific service.

Service 10 GetProcess
Precond.: $\exists serv \in SF | serv.ServiceID = ServiceID$
Input: a valid service ID
Output: a ProvidersList that contains service implementation details
Effects: —

4.4 Service Facilitator in THOMAS Framework

The THOMAS Framework is composed of OMS and SF modules. Its purpose is to obtain a product wholly independent of any internal agent platform, thus fully addressed for open systems. The THOMAS framework allows any agent to create virtual organizations with the structure and norms needed, along with the demanding and offering services required. The framework is in charge of the management of the organization structure, norms and life cycle, as well as controlling the visibility of the offered and demanded services, and the fulfillment of the conditions to use them.

In this way, the capabilities provided by the OMS and the SF are implemented as a set of independent services that can be allocated in any host, even in different ones. Services are accessible via WSDL descriptions, as common web services, or by using more elaborated communications or even negotiation processes via ACL messages. Moreover, the OMS and SF entities are implemented as autonomous agents that play the OMS and SF roles, respectively, and invoke and use these services. Services are described by means of OWL-S[1]. OWL-S is a well-known standard ontology for describing semantic services. However, this does not necessary mean that the services managed by the THOMAS framework must be typical web services, since a more general concept of service is considered here.

Regarding the platform kernel, the PK provides a unique entry point for all the external entities and redirects every message to the corresponding component: the SF for service management and the OMS for organization management. In the THOMAS framework, JADE[2] is the platform used as PK. The URL address of this platform and the methods for doing this 'first contact' situation must be public and known by the entities that try to belong to the THOMAS framework.

From a virtual organization point-of-view, all agents included in the framework must belong to an organization. Thus, THOMAS framework provides a *Main* organization in which any entity is automatically included, as well as a *general role* that allows the entity to ask for service descriptions so as to fulfill its needs. Throughout service descriptions stored in the SF, the client can be informed of the roles required to ask for any specific service or the roles needed to be able to provide a specific service inside an organization. Furthermore, the OMS takes control on several role enactment norms, such as limiting the number of entities playing a concrete role or checking the membership of an agent to a specific organization.

Next, a general description of how services have been implemented in the THOMAS framework is presented. Then, we described the implementation of the SF agent.

[1] http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/
[2] http://jade.tilab.com/

Finally, it is detailed how SF entities are encapsulated into agents to be accessible by other agents. Note that we focus on the service management and therefore, we do not provide details about the implementation of the OMS component. The implementation of the OMS is similar to the SF but for more information about its implementation we refer the reader to [33].

4.4.1 Service Implementation

THOMAS Services are implemented as semantic web services, using Axis2[3], more specifically Apache Axis2/Java as core engine for web services. Each service has a WSDL description and a semantic description in OWL-S. The WSDL document describes network services as a set of endpoints operating on messages that contain either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. The OWL-S document details the properties and capabilities of a web service in unambiguous, computer-interpretable form. The OWL-S description facilitates the automation of web service tasks, including automated service discovery, execution, composition and interoperability.

All SF and OMS services are registered in the SF, and also agent and organization services. For each service registered in the SF, there should be two OWL-S documents, one with the profile description (mandatory) and the other one with the process and grounding description. This division avoids redundant information in the SF if a service with the same profile is provided by two providers. In this case, both providers offer the service with the same IOPE[4] parameters but different implementation (i.e., they employ different process models and groundings).

[3] http://www.jaxmag.com/itr/online_artikel/psecom,id,747,nodeid,147.html
[4] input, output, preconditions and effects

4.4.2 Implementation of the SF component

The SF has to be able to manage services provided by external agents. In order facilitate the service management, semantic information is added as OWL-S descriptions for services and ontology specification in OWL for agent beliefs interchange and the specification of service parameters. Thus, tools to handle semantic information have been added to the SF.

OWL-S allows to describe a web service in terms of a *Profile*, which tells "what the service does", a *Process Model*, which tells "how the service works", and a *Grounding*, which tells "how to access the service". The *Profile* and *Process Model* are considered to be abstract specifications, in the sense that they do not specify the details of particular message formats, protocols, and network addresses by which a web service is instantiated. The role of the *Grounding* is to provide these more concrete details. For each service in the SF should be two OWL-S documents, one with the profile description, and the other with the process and grounding description. The reason to divide the service description in two files is to avoid redundant information in the SF (Figure 4.2).

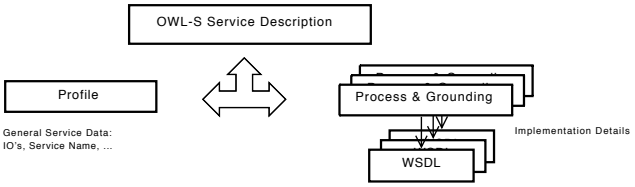


Figure 4.2: Semantic service description in OWL-S.

The SF web services manipulate semantic information in OWL using JENA. JENA[5] manages all these semantic data in OWL. Jena is a Java framework for building semantic web applications. It provides a programmatic environment for RDF, RDFS, OWL, SPARQL, and includes a rule-based inference engine. Jena provides an implementation of the RDF model interface that stores the triples persistently in a database.

[5] http://jena.sourceforge.net/ontology/index.html

Each triple is an arc in an RDF model that is called a statement. Each statement asserts a fact about a resource. A statement has three parts :

- the subject: is the resource from which the arc leaves
- the predicate: is the property that labels the arc
- the object: is the resource or literal pointed to by the arc

In Figure 4.4 and 4.3 there is an example of some statements stored in by the SF. These statements are part of a registered service *SearchCheapHotel*.

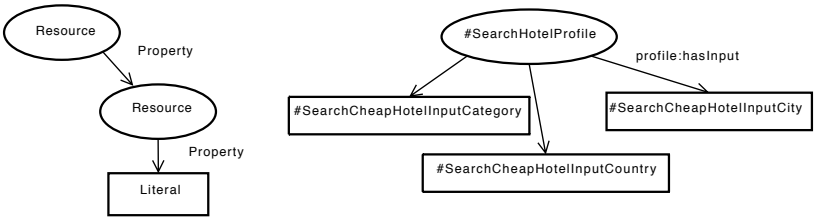


Figure 4.3: RDF graph.

```
<http://.../SearchCheapHotelProfile.owl#SearchCheapHotelProfile>;
  a       profile:Profile ;
  profile:contactInformation      mind:ProviderA ;
  profile:hasInput
   <http://.../SearchCheapHotelProfile.owl#SearchCheapHotelInputCity>;,
   <http://.../SearchCheapHotelProfile.owl#SearchCheapHotelInputCategory>;,
   <http://.../SearchCheapHotelProfile.owl#SearchCheapHotelInputCountry>;
;
  profile:hasOutput
   <http://.../SearchCheapHotelProfile.owl#SearchCheapHotelOutputHotel>;,
   <http://.../SearchCheapHotelProfile.owl#SearchCheapHotelOutputHotelCompany>;
;
  profile:serviceName "SearchCheapHotel"@en ;
  service:isPresentedBy
   <http://.../SearchCheapHotelProfile.owl#SearchCheapHotelService>;
.
```

Figure 4.4: RDF triples.

Furthermore, SPARQL[6] has been used as a query language. SPARQL is ”data-oriented” language that only queries the information held in the models; there is no inference in the query language itself. The Jena model may be 'smart' in that it provides the impression that certain triples exist by creating them on-demand, including OWL reasoning. SPARQL does not do anything other than take the description of what the application wants, in the form of a query, and returns that information, in the form of a set of bindings or an RDF graph. In Figure 4.5 there is an example of a SPARQL query about the service profile of a service with the name *”SearchCheapHotel”*.

```
PREFIX  profile: <http://www.daml.org/services/owl-s/1.1/Profile.owl#>;
SELECT  ?x
WHERE
  { ?x  profile:serviceName  "SearchCheapHotel"@en .}

Query Result
ServiceID:
http://.../SF/OWLS/SearchCheapHotelProfile.owl#SearchCheapHotelProfile
Profile http://.../SF/OWLS/SearchCheapHotelProfile.owl
```

Figure 4.5: SPARQL query about services that have in their profile the serviceName field with the value ”SearchCheapHotel”.

One of the goals of agents that play the *SF* role is providing an intelligent service discovery. The discovery of services would be seriously limited to discover single services if algorithms do not address the issue of discovering service compositions. In many situations, queries that cannot be satisfied by a single service might be frequently satisfied by composing several services. The *SearchService* provided by agents that play the *SF* role has to consider service composition when there is no a single service available that can deal with an agent or organization requirements. The task of automated service composition is to automatically sequence together services into a composition that achieve some defined goals. Several approaches consider service composition problem as a planning problem [24, 107]. An important benefit of using a planning approach is the exploitation of knowledge that has been accumu-

[6] http://www.w3.org/TR/rdf-sparql-query/

lated over years of research on the field of planning. Therefore, well known planning algorithms, techniques and tools can be used to provide efficient service compositions. For this reason, we have considered the use of planning techniques to provide service composition functionality in the agents that play the SF role in THOMAS.

DEFINITION **6** *(Planning problem) In general, a planning problem can be described as a five tuple* $< ST, S_0, \mathcal{G}$,*Acc*,$\Gamma >$*, where:*

- ST *is the set of possible states of the world,*
- S_0 *denotes the initial state of the world,*
- $\mathcal{G}$ *denotes the goal state of the world that the planning system attempts to reach,*
- *Acc is the set of actions the planner can perform in attempting to change from one state to another state in the world,*
- $\Gamma \subseteq \mathcal{G} \times Acc \times S$ *is translation relation that defines the preconditions and effects for the execution of each action.*

In the context of services, a planning problem can be seen as a service composition problem where:

- ST is the set of possible partial service compositions;
- S_0 denotes the initial information available by the requester agent;
- $\mathcal{G}$ represents the goals of a organization or agent;
- *Acc* is the set of available services;
- Γ denotes the state change function that links services outputs-inputs;

In order to describe all the components of a planning problem, PDDL language has been widely recognized as an standardized input for state-of-the-art planners. Specifically, we are going to consider the PDDL2.1 version [49]. PDDL2.1 extends PDDL with numeric and durative extensions to achieve additional expressive power.

The modeling of temporal relationships in a discretized durative action in PDDL is done by means of temporally annotated conditions and effects. All conditions and effects of durative actions are temporally annotated:

- **Conditions.** The annotation of a condition makes explicit when the associated proposition must hold:
  - at the *start* of the interval (the point when the action is applied)
  - at the *end* of the interval (the point when the final effects of the action are asserted)
  - *over* the interval from the start to the end (invariant over the duration of the action)

  Invariant conditions in a durative action are required to hold over an interval that is open at both ends (starting and ending at the end points of the action). These are expressed using the *over all*. If one wants to specify that a fact p holds in the closed interval over the duration of a durative action, then three conditions are required: *(at start p)*, *(over all p)* and *(at end p)*.

- **Effects.** The annotation of an effect makes explicit whether the effect is immediate (it happens at the start of the interval) or delayed (it happens at the end of the interval). No other time points are accessible, therefore all discrete activity takes place at the identified start and end points of the actions in the plan.

Moreover, an action can have associated a numeric duration that is represented by a variable called *duration* that represents the durative interval.

We have considered the posibility of describing services similarly to temporal actions in PDDL 2.1. Specifically, we have added temporal annotations to semantic

service description in OWL-S. OWL-S is a well-known standard ontology for describing semantic services and its usage facilitates the creation of interoperable, and open systems being possible for any entity to understand and employ these service descriptions. OWL-S pre-conditions and effects have been temporally annotated with the same labels that appear in a PDDL domain definition: *at start*, *at end* and *overall* (Figure 4.6). The OWL-S inputs and outputs are not temporally annotated, but if the service is considered as a durative action all input parameters are considered as if they were annotated with the label *at start*. The case of the outputs is similar, but the parameters are considered as if they were annotated with the label *at end*. An example of a service with temporal annotations is shown in Figures 4.6 and 4.7.

In our proposal, all the temporal annotations in preconditions and effects of a OWL-S service description are inside a PDDXML expression. PDDXML [74] is a XML dialect of PDDL that simplifies parsing, reading, and communication PDDL descriptions using SOAP. We have extended this XML language with new labels to facilitate the temporal annotation in the OWL-S service descriptions.

```
<process:hasPrecondition>
  <pddxml:PDDXML-Condition rdf:ID="PDDXML-Precondition">
   <expr:expressionBody rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    <and>
     <atStart>
      <not>
       <pred name="agentHasKnowledgeAbout">
        <param>?http://.../Packing/GetItems.owl#BookFlightFinishEvent
        </param>
       </pred>
      </not>
     </atStart>
    </and>
   </expr:expressionBody>
  </pddxml:PDDXML-Condition>
 </process:hasPrecondition>
```

Figure 4.6: Temporally annotated precondition in *BookFlight* OWL-S service description.

Although PDDL and OWL-S languages are similar, planners cannot deal directly with semantic service descriptions in OWL-S. OWL-S service description should be translated into a PDDL representation [115]. The conversion process from OWL-S

```
<Duration_param:hasLocal>
  <duration:Duration-Expression rdf:ID="PDDXML-Duration">
   <expr:expressionBody rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    <and>
     <equals>
      <variable><var type="object">?duration</var></variable>
      <constant><const type="int">4</const></constant>
     </equals>
    </and>
   </expr:expressionBody>
  </duration:Duration-Expression>
 </Duration_param:hasLocal>
```

Figure 4.7: *Duration* non-functional parameter in OWL-S service description.

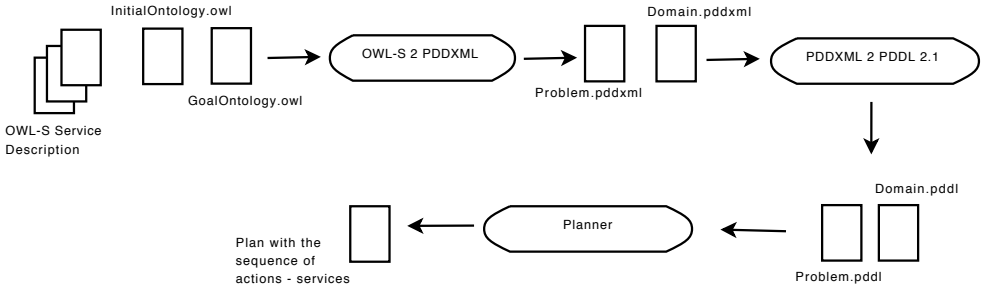


Figure 4.8: Translation process from OWL-S/OWL to PDDL 2.1.

to PDDL takes as inputs: a set of available *OWL-S temporal service descriptions*, a *domain description*, and a *query*. The *domain description* is a set of OWL individuals (i.e., facts) that are initially true (initial state). The *query* contains a set of OWL individuals that must be achieved by a plan (goals). The result of the process is a plan sequence, i.e., a composite service that satisfies the query and the temporal restrictions.

The composition process is divided in three stages (see Figure 4.8):

- *From OWL-S service descriptions and OWL facts to PDDXML.* This stage converts the domain ontology in OWL and service descriptions in OWL-S to an intermediate language in XML. This stage is an extension of the converter pre-

sented in [74]. The converter presented in [74] converts OWL-S documents with PDDXML expressions, but neither consider temporal annotations nor the duration non-functional parameter; therefore, time annotations cannot be considered in the service composition process.

This conversion process requires the transcription of types and properties from OWL to PDDL predicates as well as the mapping of services in OWL-S to actions in PDDL. Any OWL-S service *input* parameter correlates with an equally named *input* of a PDDL action, and the *hasPrecondition* service parameter can directly be transformed to the precondition of the action. The same holds for the *hasEffect* parameter. For the conversion of the *output* of an OWL-S service to PDDL, the service output is mapped to a special type of *hasEffect* parameter. This is because the service *hasEffect* condition explicitly describes how the world state will change. This is not necessarily the case for a service *hasOutput* parameter value, though it could implicitly influence the composition planning process. PDDL does not allow describing such non-physical knowledge. This problem can be solved by mapping the service output parameter X to a special type of *hasEffect* parameter. In particular, every output variable X is described in, and added to the current (physical) planning world state as a predicate in PDDL uniquely named *"agentHasKnowledgeAbout(X)"*. Similarly, each input variable Y is mapped to an input parameter Y of an PDDL action complemented by precondition predicate *"agentHasKnowledgeAbout(Y)"* (Table 4.4).

- *From PDDXML to PDDL 2.1* is a process in which a parser translates the domain and problem specification in PDDXML in an equivalent PDDL 2.1 problem and domain descriptions.

- *Planner* deals with PDDL 2.1 language and it is used to obtain, if it is possible, a sequence of durative-actions to achieve the goal state from the initial state. Any planner that deals with PDDL problems can be used (i.e., Fast-Forward, GraphPlan, LPG). The final plan represents the service composition that an-

OWL-S 1.1	PDDL 2.1
hasPrecondition parameter	precondition predicate
hasEffect parameter	effect predicate
hasInput parameter	input predicate + additional precondition agentHasKnowledgeAbout(Input param)
hasOutput parameter	effect predicate + agentHasKnowledgeAbout(Output param)

Table 4.4: Mapping between OWL-S service description and PDDL action.

swers the query.

```
(:durative-action BookFlightService
 :parameters (
?BookFlightInputDestination - object
?BookFlightInputDepartureDate - object
?BookFlightInputArrivalDate - object
?BookFlightOutputBookCode - object
?BookFlightFinishEvent - object )
 :duration ( = ?duration 4 )
 :precondition (and
            (at start (agentHasKnowledgeAbout ?BookFlightInputDestination ))
            (at start (Destination ?BookFlightInputDestination ))
            (at start (agentHasKnowledgeAbout ?BookFlightInputDepartureDate ))
            (at start (DepartureDate ?BookFlightInputDepartureDate ))
            (at start (agentHasKnowledgeAbout ?BookFlightInputArrivalDate ))
            (at start (ArrivalDate ?BookFlightInputArrivalDate ))
            (at start (not(agentHasKnowledgeAbout ?BookFlightOutputBookCode)))
            (at start (not(agentHasKnowledgeAbout ?BookFlightFinishEvent )))
            )
  :effect (and
          (at end (agentHasKnowledgeAbout ?BookFlightOutputBookCode ))
          (at end (BookCode ?BookFlightOutputBookCode ))
          (at end (agentHasKnowledgeAbout ?BookFlightFinishEvent ))
          (at end (FinishEvent ?BookFlightFinishEvent ))
          )
)
```

Figure 4.9: OWL-S temporally annotated service *BookFlightService* translated into a Durative-action in PDDL.

4.4.3 Agent Implementation

Both the OMS and SF entities have been developed as JADE agents that play the roles *OMS* and *SF* defined in Definitions 1 and 2, and offer the services associated to these roles. The logic of these agents has been implemented using the OWL-S API provided by Mindswap[7]. This OWL-S API employs a Java API for programming access to read, execute and write OWL-S service descriptions. When the SF agent (or the OMS agent) receives a FIPA request message from a client, it employs the OWL-S API to access the service description in OWL-S and execute the correspondent web service. As an example, Figure 4.10 shows the interaction between a client agent, the SF agent, and the SF web services.

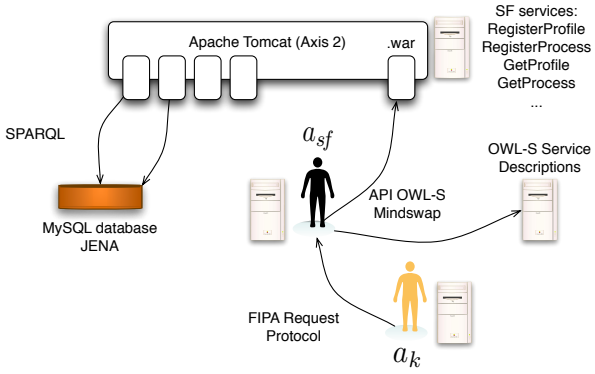


Figure 4.10: SF agent in THOMAS Framework.

A service execution implies accessing to the process information included in the service description. The service process description should have a valid grounding specification in order to invoke the service successfully.

Along this section, the implementation features of the THOMAS framework have been described in detail. We have focused on the implementation of the SF component. The SF component is an intermediary agent for accessing semantic web

[7] http://www.mindswap.org/2004/owl-s/api/

services that deal with the management of services offered by agents in organizations. The next section illustrates a case study which describes how an external agent, which has been designed independently of the THOMAS framework, makes use of both THOMAS and registered functionalities.

4.5 Application Example

In order to illustrate the performance of the THOMAS framework with greater detail, a case-study example for making flight and hotel arrangements [40, 123] has been implemented. The *Travel Agency* example is an application that facilitates the interconnection between clients (individuals, companies, travel agencies) and providers (hotel chains, airlines); delimiting services that each one can offer and/or request.

A simple demo example of a travel agency system can be found together with an available prototype of the THOMAS architecture[8]. Following, a description of the organizational structure of the *TravelAgency* organization is explained, and a dynamical usage of the organization is detailed.

Organization Structure. This case study is modeled as an organization (*TravelAgency*), in which two kinds of roles can interact: *customer* and *provider*. The *Customer* role requests travel services of the system, i.e., it can request hotel or flight search services, booking services for hotel rooms, or flight seats and payment services. The *Provider* role is in charge of performing these travel services. Thus, a *provider* agent offers hotel or flight search services, and can also offer booking hotel rooms or flight seats. The *TravelAgency* organization offers three services: *SearchTravel*, *Booking* and *Payment* service. In addition to service identification, designers should provide an OWL-S description of each service offered inside the *TravelAgency*. As an example, Table 4.5 contains the service profile corresponding to *SearchTravel* service.

[8] http://www.dsic.upv.es/users/ia/sma/tools/Thomas/index.html

Service		**Description**		**UnitID**
SearchTravel		Search for travel information		TravelAgency
ClientRole	**ProviderRole**	**Inputs**	**Outputs**	
Customer	TravelOrganizer	city:string country:string	[city ok] company:string location:string price:float	[not in city] error

Table 4.5: Profile of the service *TravelSearch*.

Dynamical Usage. The system dynamics are shown through the specification of two scenarios. The first scenario shows how an agent registers a new implementation of a service in an organization (see Figure 4.11). The second scenario shows the interactions carried out by an external agent to ask for a service (see Figure 4.12). These examples illustrate how the employment of the web service standards for both providing and publishing services allow agents to discover and make use of both agent and THOMAS functionalities.

In the first scenario, agent a_k asks the SF agent for the list of services registered in the platform that are related to *Travel*. Agent a_k receives from the SF a list with the services. Then, agent a_k asks for the profile of the service with highest rank. Based on the profile, agent a_l knows the the inputs, outputs, preconditions and effects of the service and the role that must be acquired to be a provider of this service. Agent a_k provides a service with the same profile and it wants to register its implementation of the service. To do that, a_k acquires the role *provider*. Then, agent a_k asks the SF agent for the implementation details of the service *SearchTravel*. If the implementation provided by agent a_k is not equal to any of the implementations already registered, then the agent a_k registers its process. Otherwise, a_k adds itself as a provider associated to an existing implementation.

In the second scenario, an external agent joins the THOMAS framework, registers itself as a service client and carries out a service request (Figure 4.12). The client agent (a_m) requests the *SearchService* to the SF component so as to find services of

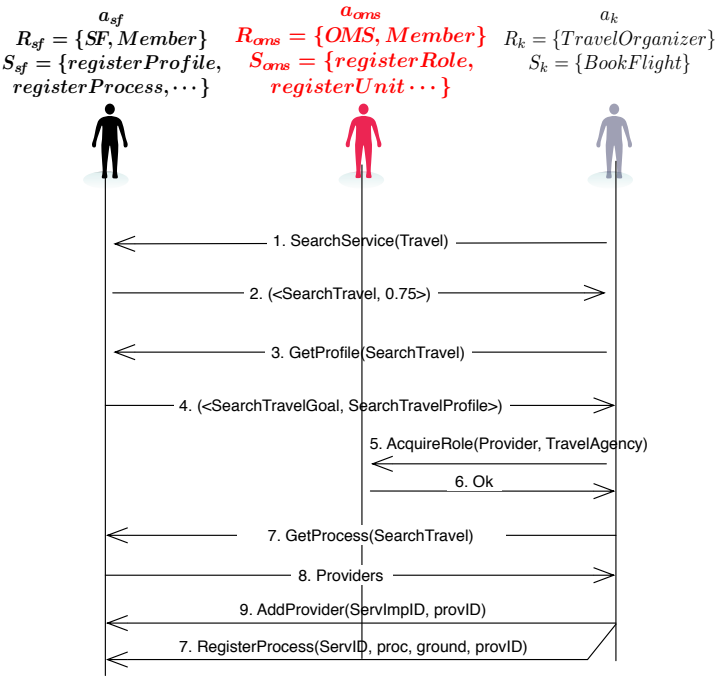


Figure 4.11: Provider registering a service in THOMAS Framework.

its interest (message 1). The result of this service is shown in message 2. Then, a_m asks the SF agent for the profile of the service it is interested in. Agent a_k analyzes the profile and then it knows that it must acquire the *Customer* role to demand this service (messages 3-6). Once a_k plays this customer role, it employs the *SearchProvider* service in order to know who the service providers are and how this service can be requested (messages 9 and 10). As shown in message 10, one implementation of the *SearchTravel* service has been previously registered. This implementation has one provider (a_k). Agent a_m makes a service petition to a_k agent, so then according to this service process, a_m sends a message for requesting this *SearchTravel* service to a_k agent (messages 9 and 10).

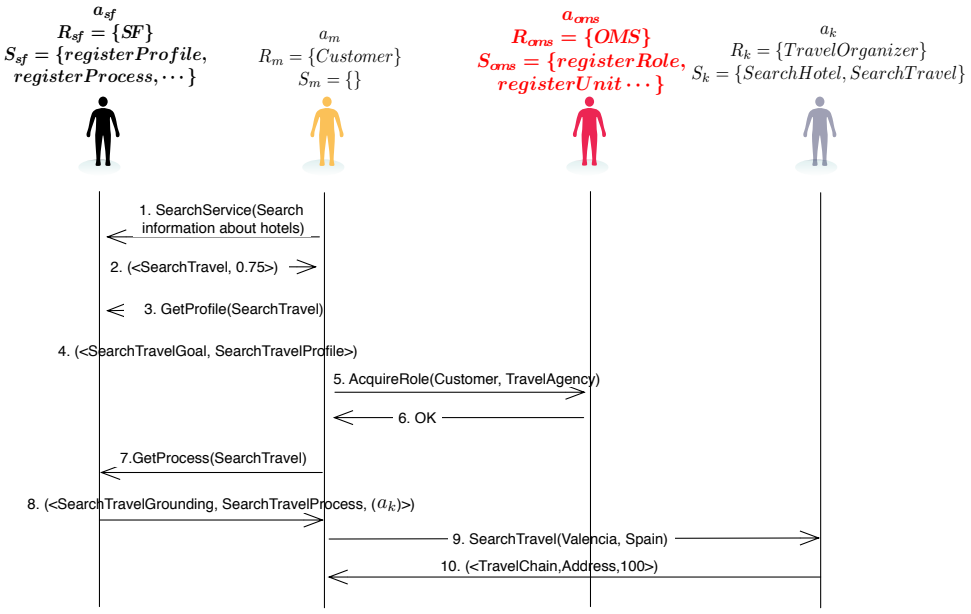


Figure 4.12: Client requesting a service in THOMAS Framework.

4.6 Conclusions

In this chapter, we have described the THOMAS architecture that integrates agents and services, and provides mechanisms that facilitate the management of virtual organizations and the services provided by agents that are part of these organizations. The SF and OMS roles present in this architecture describe the functionality required to provide a suitable management of VOs. Based on this architecture a new service-oriented execution framework for supporting the development of real VOs, named THOMAS framework, has been presented. This framework is suitable for the development of large scale open multi-agent systems and capable of managing the complexity, uncertainty and dynamic features of virtual organizations in open systems in an efficient way.

This framework extends the previous proposals taking as a reference the challenging

development of open systems. Therefore, the main contributions of the THOMAS framework are:

- it gives support to *virtual organizations* as a social abstraction for coordinating the autonomous behaviors of agents;
- system functionalities are described and provided as *meta-services* in order to make possible the interaction among heterogeneous entities in a standardized way;
- it provides *discovering* and *composition* services for allowing external agents, which have been designed independently of the THOMAS framework, to participate inside it;

A case study, which illustrates how an external agent is capable of discovering and employing functionalities in THOMAS, is also provided. All this work has been included in an available prototype of the THOMAS abstract architecture.

CHAPTER 5

Decentralized Service Management in Service-Oriented MAS

5.1 Introduction

Service management and specifically service discovery is a key issue in facilitating the cooperation and the goal fulfillment of agents that are organized in virtual organizations (VOs). In these VOs, the life-cycle of the organizations and the available services provided by agents are centralized and controlled in a set of entities. Nevertheless, there are Service-Oriented MAS where there is no central control on how agents should be connected or disconnected and there is no maintenance of system

structure. These features provide more flexibility and adaptability to the system. However, service discovery becomes more complicated. Agents usually have no global knowledge about the system structure —they only know information about their direct neighbors. Therefore, agents need the collaboration of the rest of the agents in the system to succeed in the service discovery process. Thus, locating a service efficiently is considered to be one of the most important challenges in this area [14].

In these environments, centralized mechanisms such as registries or middle-agents are not efficient in dealing with this challenge. Weaknesses such as bottlenecks, lack of coordination, outdated data, or the need of huge amounts of memory to store information about the agents' services make centralized approaches unsuitable for coping with dynamic system requirements. Moreover, one of the most important drawbacks is that these mechanisms rely on global knowledge and, this global knowledge is usually not present in open Service-Oriented MAS. Hence, decentralized service discovery mechanisms are required in these systems.

One of the areas of interest that has structures and search strategies for dealing with decentralized service discovery is the area of Complex Networks [16]. Complex Networks have new, less rigid structures that are inspired in social, biological, or technological networks and algorithms that facilitate the search in distributed environments. This area proposes models to create efficient structures in a self-organized way without the supervision of a central authority. Moreover, in some of these structures, a target can be found in just a few steps and considering only local information [141, 71, 139, 4, 70]. Some of these models take into account properties that are present in human societies as a criteria for establishing links. One of these properties is homophily [89].

Homophily, also known as assortativity [95, 96], is one of the most salient properties present in complex networks [89, 26, 12]. The term 'homophily' was introduced by Lazarsfeld and Merton [77] in 1954. The idea behind this concept is that individuals tend to interact and establish links with similar individuals. Therefore, homophily es-

tablishes the proportion in which two individuals are similar based on a set of social dimensions. These social dimensions are attributes such as religion, age, or education. Therefore, in a complex network model based on homophily, an individual has a higher probability of being connected to a more similar individual than to a dissimilar one. This criterion to establish links between individuals creates structures that facilitate the location task [139, 128, 70, 34]. For this reason, homophily could be considered as a self-organizing principle to generate searchable structures.

In this chapter, we present a decentralized service management system for Service-Oriented MAS where homophily has been introduced as a self-organizing criterion to create the social structure of the system and as a criterion to guide the service discovery process. The structure is a network based on preferences where agents create links with other agents by considering their homophily based on two social dimensions: services and organizational roles. We also propose an algorithm that allows agents to locate services offered by other agents using only local information, without any centralized service repository or directory. This algorithm offers good performance not only in networks based on homophily but also in other network structures.

The proposal described in this chapter attempts to improve previous approaches in several ways. First, as a decentralized system, all the agents are considered to be equal and they only consider local information in the service discovery process and to establish links. Therefore, the system provides robustness, scalability, and adaptability. Second, the system is self-organized based on homophily between agents and does not need an initial period to establish its structure. Progressively, each agent that joins the system establishes links with agents that share features such as the organizational role or the services offered. Third, each agent only maintains a local view of the services it offers and who are its neighbors, and it does not maintain information about routes that could change frequently in highly dynamic environments. Fourth, in our system, the algorithm for the service discovery process is not based on previous information or statistics that require a training period in order to be reliable. The

algorithm is based on similarity between agents, and this similarity is calculated considering the semantic descriptions of the agents and not just keywords or pre-defined categories.

This chapter is structured as follows. Section 5.2 describes the service discovery scenario. Section 5.3 explains how we have introduced the concept of homophily in the service discovery context. Section 5.4 describes the process that creates a network structure that facilitates the service discovery. Section 5.5 explains the service discovery process. Section 5.6 presents the results of the experiments to evaluate the network structure and the discovery process. Finally, Section 5.7 presents conclusions and final remarks.

5.2 Decentralized Service Discovery Scenario

To illustrate the context where decentralized service discovery is applied, let us present a service discovery scenario where the discovery process is described (see Figure 5.1). Consider a network of agents as a form of autonomic cloud computing system. This network contains different groups of specialized computing systems as part of an overlaying network where semantic web services are provided by software agents. Agents play an organizational role that defines the type of services they offer. Agents only have information about their direct neighbors with which they have a connection with and they do not know about the other agents that are part of the system, the number of agents or the system structure. Neither, there are the figure of an intermediator or central registry that has a global and complete vision of the hole system or part of it.

5.2.1 Network Creation

The structural relations between these agents have been established taking the homophily criterion into account. The idea behind the homophily concept is that in-

dividuals tend to interact and establish links with similar individuals through a set of social dimensions. In the context of Service-Oriented MAS, two agents are considered similar if they play similar roles and offer similar services. For each pair of agents, the higher the homophily value is, the more similar the agents are. Homophily is a probabilistic concept; therefore, agents have a higher probability of establishing connections with similar agents than with dissimilar ones.

In Figure 5.1, agent a_i has connections with agents a_k, and a_j, which play similar roles and offer similar services, and a_n, which plays a dissimilar role and offers a dissimilar service. Note that agents that play similar roles are represented in Figure 5.1 with similar colors.

5.2.2 Discovery Process

Agents, in some situations, should interact with each other to achieve a task that they cannot afford to do individually since they are not specialized in that area or because the task is too complex to be carried out by a single agent.

Agent a_i offers the service $bookHotel$; however, in order to achieve one of its goals, it needs to locate an agent that offers a $rentalCar$ service and plays $Transport$ role. In that moment, agent a_i creates a query $q = \{a_i, rentalCar, Transport, TTL, \varepsilon\}$ that consists of the identifier of the agent that creates the query, the required semantic service description, the organizational role that the target agent should play, the Time To Live (TTL) that limits the number of steps that the query can be forwarded, and the threshold ε that establishes when a service it is similar enough to finish the discovery process. If the query exceeds the TTL, it is considered to be a failure of the service discovery process. Otherwise, the query is forwarded to one of the neighbors. It is assumed that all the agents are collaborative and follow the same criterion to forward the queries.

In the scenario of Figure 5.1, agent a_i should choose one of its neighbors, a_n,a_j, or a_k, to forward the query q. In order to select the most promising neighbor, the

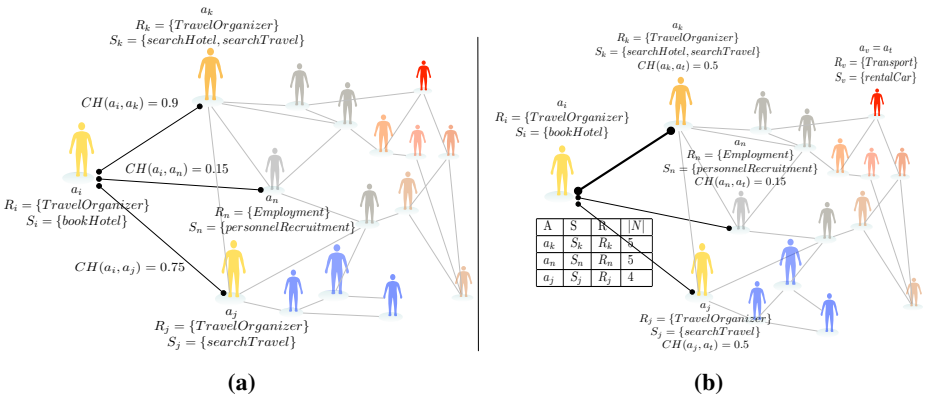


Figure 5.1: An example of decentralized service discovery system. (a) Agent a_i establishes a link with two similar agents a_k and a_j and with a dissimilar one a_n; (b) Agent a_i only knows its direct neighbors a_k, a_j, and a_n. If a_i needs to locate a service (i.e., $rentalCar$), it will forward the query to its most promising neighbor (i.e., a_k) based on the homophily between the neighbor and the target agent (i.e., a_t) that should provide the required service and the degree of the neighbor.

agent a_i considers: (i) the homophily between the neighbors and an hypothetical unknown target agent a_t that offers the service and plays the role specified in the query q; and (ii) the degree of connection of the neighbors. Assuming the values of choice homophily that appear in Figure 5.1, agent a_i sends the query to the most promising agent (i.e., agent a_k). This process is repeated until the similarity between a local service of an agent and the service in the query is over a certain threshold, or the query exceeds the TTL. In the described scenario, the process ends when the query arrives to agent a_v that is similar to the hypothetical target agent a_t that a_i was looking for.

5.3 Introducing Homophily in Service-Oriented MAS

Agents in this model are self-organized considering a social feature called *choice* homophily (CH) [89]. This type of homophily is the factor that allows agents to establish links with similar agents based on a set of social dimensions and guides

the decentralized discovery process. Choice homophily is subdivided into two types: (i) *value homophily*, which is based on the similarity of shared attributes (such as gender, age, geographical location, and so on); and (ii) the *status homophily*, which is related to the formal or informal status similarity of the individuals (social status, status within an organization, or professional degree).

In this section, we focus on the formal definition of *choice homophily* in the context of Service-Oriented MAS and how it is included in the system as a criterion to self-organize the structure and guide the service discovery process. As stated above, choice homophily is divided into two types: *value* and *status*. If these two concepts are matched with the agency-related concepts, value homophily represents the individual characteristics of the agent (which are the services the agent offers), whereas status homophily can be identified with the semantic description of the role that an agent plays.

5.3.1 Value Homophily

The *value homophily* function $H_v(S_i, S_j)$ calculates the degree of matching between two sets of services, where S_i and S_j are the sets of services provided by the agents a_i and a_j, respectively. We consider each set of services S_i (or S_j) to be composed by a set of semantic concepts that can be classified in: Inputs (I_i), Outputs (O_i), Preconditions (P_i), and Effects (Ef_i).

To generalize, the level of matching between two sets of semantic concepts, C_i and C_j, is calculated through a *bipartite matching graph* (see Figure 5.2). Let $G = (C_i, C_j, E)$ be a complete, weighted bipartite graph that links each concept $c_i \in C_i$ to each concept $c_j \in C_j$, $e_{ij} = (c_i, c_j) \in E$, and let E represent the edges established in the graph $E = C_i \times C_j$. The term ω_{ij} represents the weight associated to the arc $e_i = (c_i, c_j) \in E$ between c_i and c_j as the semantic similarity between those concepts. Four degrees of matching can be identified: *exact, subsumes, plug-in*, and *fail* [104]. The match is considered to be:

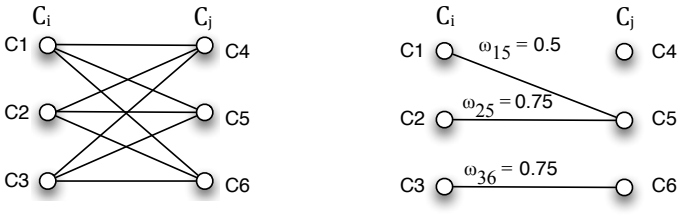


Table with the ω_{ij} values

	C4	C5	C6
C1	0.25	0.5	0.25
C2	0	0.75	0
C3	0.5	0.25	0.75

$W_{G'} = (0.5+0.75+0.75)/3 = 0.67$

Figure 5.2: (Left) Full connected weighted bipartite graph G, and (Right) resulting maximum weighted matching relaxed bipartite graph G'.

- *exact* if $c_1 \in C_i$ is equivalent to $c_2 \in C_j$ ($c_1 \equiv c_2$);
- it is *subsumes* if c_1 subsumes c_2 ($c_1 \sqsupset c_2$);
- it is *plug-in* if c_1 is subsumed by c_2 ($c_1 \sqsubset c_2$);
- and it is *fail*, otherwise.

For simplicity, we have considered these four degrees of matching but other degrees could be considered [73]. A value in the interval $[0, 1]$ is assigned to each degree of matching, where 1 represents an exact matching among the terms. The best match among concepts is obtained by calculating the *maximum weighted bipartite matching*, $G' = (C_i, C_j, E')$, where $E' \subseteq E$ are the edges that have the maximal value. The graph G' is a relaxed bipartite graph because not all the concepts from C_j have to be connected to a concept in C_i; therefore, two concepts from C_i can share a concept from C_j.

Specifically, to calculate the *value homophily*, four bipartite graphs are defined, (one for each one of the components of services present in the sets S_i and S_j): Inputs

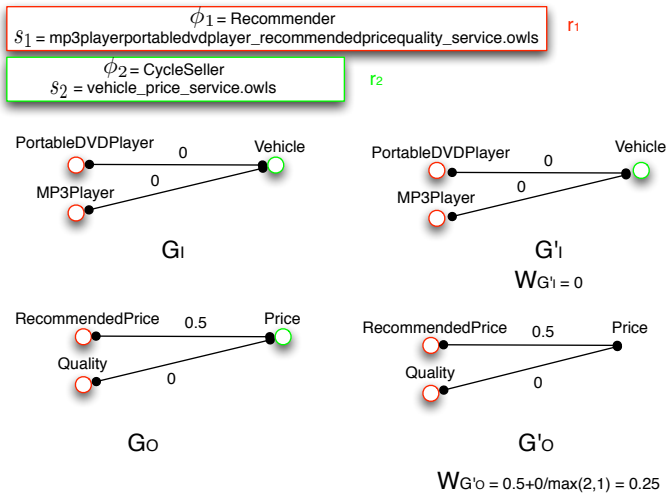


Figure 5.3: Example of $W_{G'_I}$ (top) and $W_{G'_0}$ calculation (bottom) between two agents that offer one service each.

(I_i, I_j), Outputs (O_i, O_j), Preconditions (P_i, P_j), and Effects (Ef_i, Ef_j). Let's explain the case of the inputs. The rest of the components are treated in the same way.

Let $\overline{I_i} = \bigcup_{s_i \in S_i} I_i$ be the set formed by all the inputs of all the services s_i of the agent a_i; Let $G_I = (\overline{I_i}, \overline{I_j}, E)$ be the weighted bipartite graph among the inputs of all the services S_i and S_j provided by agents a_i and a_j; and let $G'_I = (\overline{I_i}, \overline{I_j}, E')$ be the maximum weighted relaxed bipartite matching. Then $W_{G'_I}$ is defined as the normalized total weight of the maximum relaxed bipartite graph G'_I.

$$W_{G'_I} = \frac{\sum_{\omega_{ij} \in E'_I} \omega_{ij}}{\max\left(|\overline{I_i}|, |\overline{I_j}|\right)} \tag{5.1}$$

$W_{G'_O}$, $W_{G'_P}$, and $W_{G'_{Ef}}$ are similarly defined for outputs, preconditions, and effects, respectively.

DEFINITION **7** *The value homophily between two agents a_i and a_j is defined as*

$$H_v(S_i, S_j) = \alpha \left[\beta * W_{G'_I} + (1-\beta) W_{G'_O}\right] + (1-\alpha)\left[\beta * W_{G'_P} + (1-\beta) W_{G'_E}\right] =$$
$$= \alpha \left[\beta \frac{\sum_{w_{ij} \in E'_I} w_{ij}}{\max |\overline{I_i}|, |\overline{I_j}|} + (1-\beta) \frac{\sum_{w_{ij} \in E'_O} w_{ij}}{\max |\overline{O_i}|, |\overline{O_j}|}\right] +$$
$$+(1-\alpha) \left[\beta \frac{\sum_{w_{ij} \in E'_P} w_{ij}}{\max |\overline{P_i}|, |\overline{P_j}|} + (1-\beta) \frac{\sum_{w_{ij} \in E'_{Ef}} w_{ij}}{\max |\overline{Ef_i}|, |\overline{Ef_j}|}\right]$$

The parameters α and β assign different weights to the components of the formula. The adjustment of $\alpha, \beta \in [0, 1]$ allows varying how the parameters of the service are considered in the calculation of value homophily. The α parameter controls a data-driven homophily calculation (inputs and outputs) or a goal-driven homophily calculation (preconditions and effects). The β parameter determines the importance of the intakes (inputs and preconditions) or the consequences (outputs and effects) in the homophily calculation.

Let's see an example of how value homophily is calculated among two agents, a_1 and a_2 (see Figure 5.3). Agent a_1 has a set of services S_1 that contains a service s_1 (mp3playerportabledvdplayer_recommendedpricequality_service.owls) that has a set of inputs (I_1 = {PortableDVDPlayer, MP3Player}) and a set of outputs (O_1 = {RecommendedPrice, Quality}). Agent a_2 has a set of services S_2 that contains a service s_2 (vehicle_price_service.owls) that has a set of inputs (I_2 = {Vehicle}) and a set of outputs (O_2 = {Price}). In order to calculate the value homophily, a bipartite graph is created for the inputs G_I and for the outputs G_O. The similarity between the concepts from sets I_1 and I_2 (similarly for concepts from O_1 and O_2) labels the arcs of the graph. In this case, the concepts from I_1 only have one possible matching since there is only one input in the set I_2. Once the maximum bipartite graphs for inputs and outputs are built (G'_I and G'_O), their weights are calculated for inputs and outputs ($W_{G'_I}$,$W_{G'_O}$). To calculate the value homophily we instantiate the parameters

α=1 (which means that the services only have inputs and outputs), and β = 0.5 (which means that inputs and outputs have the same importance). We replace the values of $W_{G'_I}$ and $W_{G'_O}$ in Definition 7, and we obtain 0.125 for the value homophily between agents a_1 and a_2.

$$H_v(S_1, S_2) = 1\,[0.5 * 0 + (1 - 0.5)0.25] = 0.125 \tag{5.2}$$

5.3.2 Status Homophily

The *status homophily* $H_s(R_i, R_j)$ in the system calculates the best match between the set of roles R_i and R_j played by the agents a_i and a_j. The match between two individual roles $r_i \in R_i$ and $r_j \in R_j$ is based on the distance between the semantic concepts ϕ_i and ϕ_j. The function presented by [50] is used to calculate the distance.

DEFINITION **8** *Status homophily between two agents a_i and a_j is defined as the maximum degree of match between the concepts ϕ_i and ϕ_j that describe the roles $r_i \in R_i$ and $r_j \in R_j$ for all possible pairs (r_i, r_j).*

$$H_s(R_i, R_j) = \max_{r_i \in R_i, r_j \in R_j} rmatch(\phi_i, \phi_j)$$

where

$$rmatch(\phi_i, \phi_j) = \begin{cases} 1 & \textit{if path length = 0} \\ \delta_{\phi_i\phi_j} \cdot e^{-\lambda(pl_{\phi_i\phi_j} + cp_{\phi_i\phi_j})} & \textit{if } \phi_i \textit{ and } \phi_j \textit{ are not siblings} \\ \delta_{\phi_i\phi_j} \cdot e^{-\lambda(pl_{\phi_i\phi_j} + cp_{\phi_i\phi_j}) - d_{\phi_i\phi_j}} & \textit{if } \phi_i \textit{ and } \phi_j \textit{ are siblings} \end{cases}$$

and

$$\delta_{\phi_i\phi_j} = \frac{e^{\gamma dp_{\phi_i\phi_j}} - e^{-\gamma dp_{\phi_i\phi_j}}}{e^{\gamma dp_{\phi_i\phi_j}} + e^{-\gamma dp_{\phi_i\phi_j}}}$$

The status homophily $H_s(R_i, R_j)$ takes into account the following:

- $pl_{\phi_i\phi_j}$ the shortest path length between ϕ_i and ϕ_j in an organizational ontology;
- $dp_{\phi_i\phi_j}$ the depth of the roles in the ontology;
- $d_{\phi_i\phi_j}$ the number of the sibling nodes of each role;
- $cp_{\phi_i\phi_j}$ the relationship 'parent-child' between roles;
- λ and γ are parameters that control the influence of path length and depth, respectively.

The value obtained in the calculation of $H_s(R_i, R_j)$ ranges in the interval [0,1], where 1 indicates that the roles are the same.

Let's see the calculation of status homophily between the agents a_1 and a_2 from the previous example (see Figure 5.3). The set of roles R_1 of a_1 is composed by the role r_1 (ϕ_1 = *CycleSeller*). The set of roles R_2 of a_2 is composed by the role r_2 (ϕ_2 = *Recommender*). The calculation of this type of homophily is based on the organization structure that is shown in Figure 5.4. The roles r_1 and r_2 are depicted with a thick line. The figure shows the organizational roles and the values of the parameters involved in the calculation of $rmatch(\phi_1, \phi_2)$. If we replace the value of the structural parameters and the parameters that control the influence of the path length $\lambda = 0.3$ and depth $\gamma = 1$ in the formula of $rmatch$, we obtain $rmatch(\phi_1, \phi_2) = 0.04$ and $H_s = 0.04$ (see Equation 5.3).

$$H_s(R_1, R_2) = \delta_{\phi_1\phi_2} \cdot e^{(-0.3(7+3))} = \delta_{\phi_1\phi_2} \cdot e^{-3} = 0.0497 \cdot 0.82 = 0.04 \tag{5.3}$$

$$\delta_{\phi_1\phi_2} = \frac{e^{(4/6+3/6)} - e^{(-4/6+3/6)}}{e^{(4/6+3/6)} + e^{(-4/6+3/6)}} = \frac{e^{1.16} - e^{0.16}}{e^{1.16} + e^{0.16}} = \frac{3.21 - 0.31}{3.21 + 0.31} = \frac{2.89}{3.52} = 0.82$$

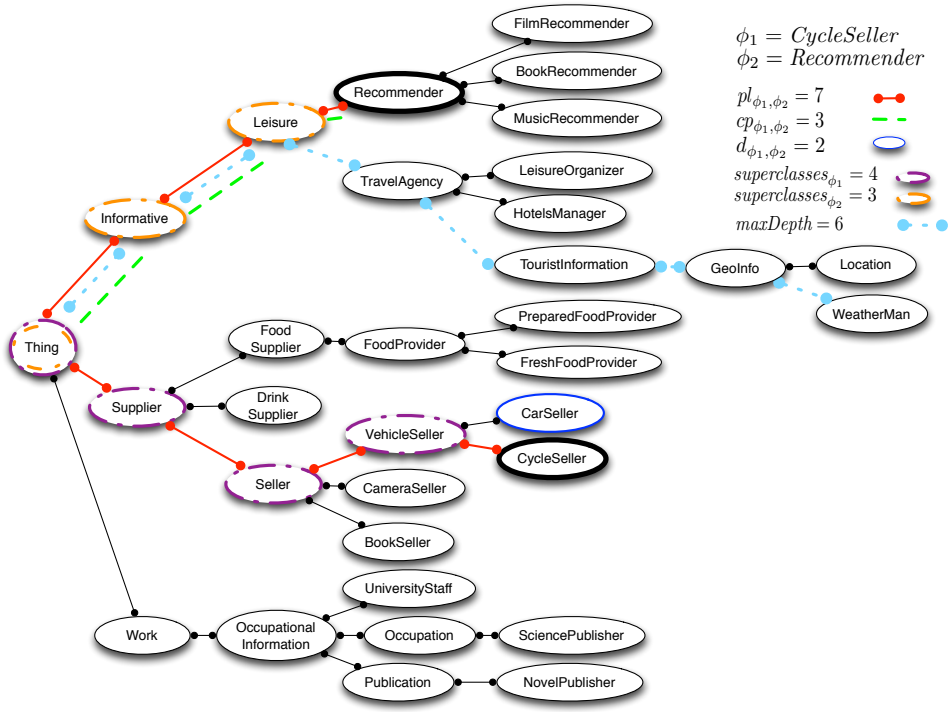


Figure 5.4: Partial view of the organizational structure that contains the semantic concepts that define the roles present in the system.

5.3.3 Choice Homophily

Once the Value and Status homophily have been defined, we can define the Choice Homophily between two agents as the linear combination of both.

DEFINITION **9** *Choice homophily between two agents $a_i, a_j \in A$ in the system is defined as the linear combination of value and status homophily,*

$$CH(a_i, a_j) = (1 - \varphi) * H_v(S_i, S_j) + \varphi * H_s(R_i, R_j)$$

The φ parameter regulates the importance of the influence of services (value homophily) or roles (status homophily) in the total homophily of the agent with another agent.

Finally, once the value and status homophily are calculated (see Equations 5.2 and 5.3), the calculation of choice homophily between agents a_1 and a_2 is shown in Equation 5.4. In this equation, the value of φ is 0.75, which means that status homophily has more influence in the final homophily between agents.

$$CH(a_1, a_2) = (1 - 0.75) * 0.125 + 0.75 * 0.04 = 0.061 \tag{5.4}$$

5.4 Community Creation based on Homophily

Choice homophily establishes a measure of semantic similarity between two agents. This similarity measure is taken into account by agents when they decide to establish a link with other agent. Here we present a growing creation process of the network and how the consideration of homophily generates a self-organized network structure based on homophily with an exponential degree of connection distribution. The self-organization based on homophily provides short paths in the discovery process. The exponential degree of connection distribution provides a resilient structure to deliberate attacks.

5.4.1 Join Protocol

Each agent that is part of the system is considered an entry point. If an external provider agent a_i wants to get into the system, it follows the protocol shown in Figure 5.5 (see Algorithm 1):

- Agent a_i should know at least one agent a_j already present in the system. Agent a_i sends a request to agent a_j to be part of the system.

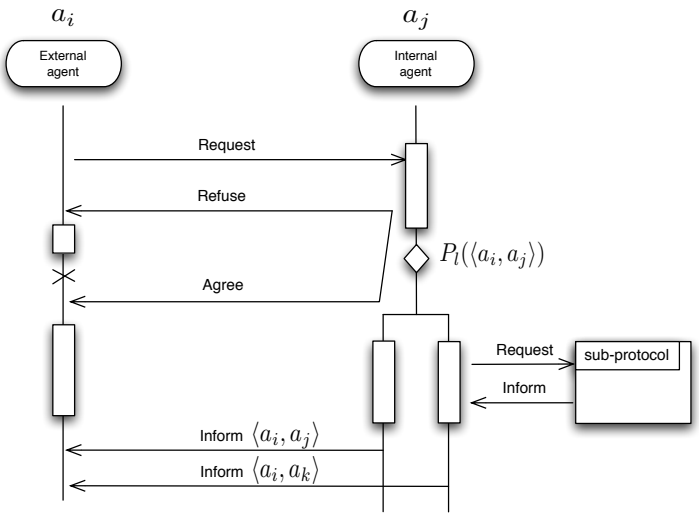


Figure 5.5: Access protocol for new agents to get into the system.

- If a_j sends a_i a refuse message the interaction finishes. Otherwise, a_j allows a_i to get into the system and it sends a_i an agree message. This means that a_j, based on the choice homophily between them, is going to consider the establishment of a link with a_i. The probability P_l of establishing a connection between agent a_i and agent a_j is

$$P_l(\langle a_i, a_j \rangle) = \left(\frac{1 - CH(a_i, a_j)}{\rho} \right)^{-r} \tag{5.5}$$

  which considers the *choice homophily* between the agents. To obtain the probability distribution, the choice homophily between two agents should be divided by an appropriate constant ρ that indicates the degree of precision to consider two agents equal. The r parameter is a homophily regulator. When r is zero, the system shows no homophily (i.e, agents are not grouped by similar services). As r grows, links tend to connect agents with more similar services. Basically, r makes the system create communities with similar services [71].

- If agent a_j decides to establish a link with a_i, it sends a inform message to a_i with the link information ($\langle a_i, a_j \rangle$). Otherwise, a_j forwards the request to one of its neighbors a_k randomly selected. The process is repeated until agent a_i receives an inform message with its neighbor ($\langle a_i, a_k \rangle$) and establishes a connection with it. The number of connections that an agent establishes is predefined by the system. Note that the link establishment process uses a random walk strategy and a probability based on homophily to find neighbors. The reason to use this random strategy, instead of a strategy based only on homophily criterion, is to give new agents the chance of establishing links not only with similar agents, but also with dissimilar ones. Links between dissimilar agents allow agents to locate other agents communities in a few steps.

Algorithm 1 Function that describes how an agent a_i enters into the system.

```
1: function JoinSystem(a_i, a_j)
2: if checkCondition then
3:     if P_l(⟨a_i, a_j⟩) > random(0, 1) then
4:         N_j ← N_j ∪ a_i
5:         inform(a_i, a_j)
6:     else
7:         a_k ← random(N_j)
8:         while P_l(⟨a_i, a_k⟩) < random(0, 1) do
9:             a_k ← random(N_k)
10:        end while
11:        N_k ← N_k ∪ a_i
12:        inform(a_i, a_k)
13:    end if
14: else
15:    refuse(a_i)
16: end if
17: end function
```

5.4.2 Structural Properties

Agents have a greater probability of establishing connections with other agents if they provide similar services (value homophily) and play similar roles (status homophily) in the system. As a result of this behavior, communities of similar agents are created

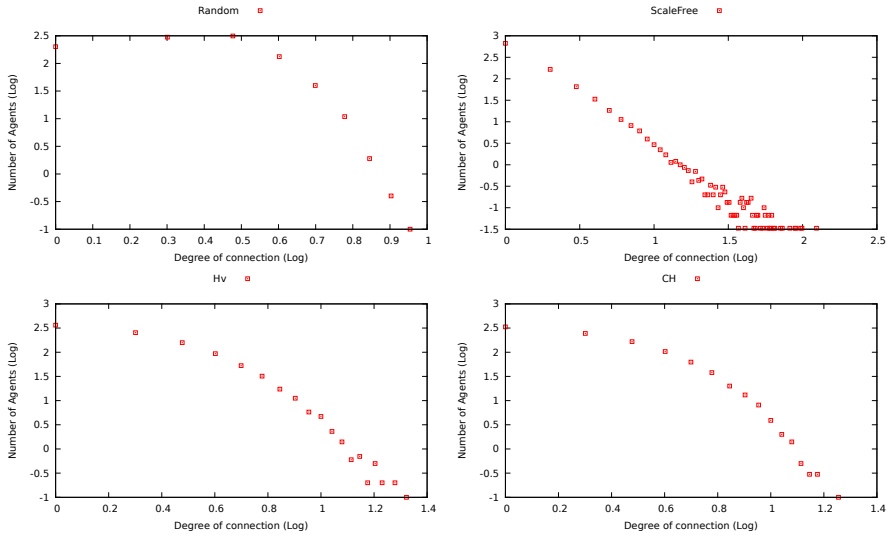


Figure 5.6: Degree of connection distribution in different type of networks: Random networks, Scale-Free networks, networks based on value homophily H_v, and networks based on choice homophily CH.

in a decentralized way. The resulting system structure is a network based on preferences, which grows according to a simple self-organized process. The construction process of a growing network ensures that the oldest nodes have a higher probability of receiving new links than the newest ones. Therefore, the total number of neighbors an agent has will depend on its age. The average degree of connection of a network built following this process follows an exponential distribution [43] (see Figure 5.6).

Because the homophily condition is a probability function, it allows new agents not only to establish 'direct connections' between agents with similar attributes (services and roles), but also between agents that are not similar. These connections allow agents to interconnect communities and locate other agents efficiently in a few steps by using only local information [71].

An example of the resulting structures is shown in Figures 5.7 and 5.8. The networks of these figures represent the structure of a system with 1,000 agents. Each node of the network represents one agent that plays one role and offers one service. The

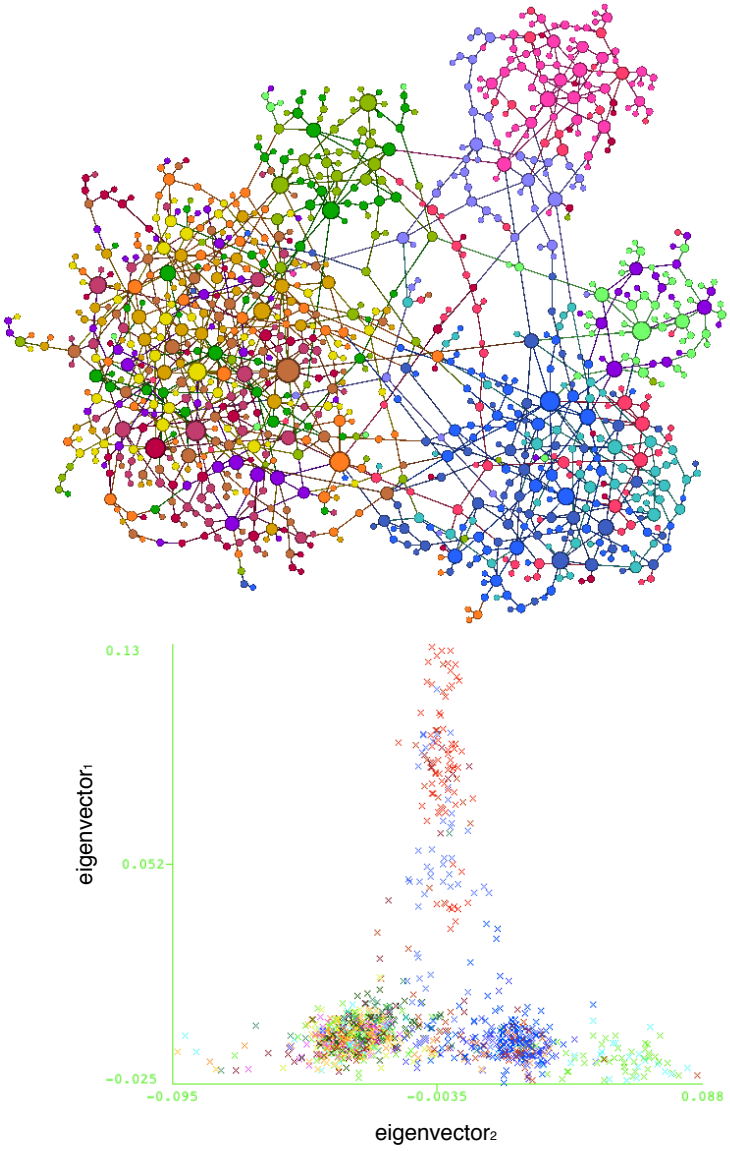


Figure 5.7: System with 1000 agents, 16 roles, and one semantic service per agent. Each node color reflects one role. Similar colors reflect that the roles are close to each other in the organizational ontology (see Figure 5.4). (Top) The network structure reflects the effects of introducing choice homophily to the system with the parameter $\varphi = 0$. (Bottom) Communities obtained through a clustering algorithm based on eigenvectors of the adjacency matrix [41].

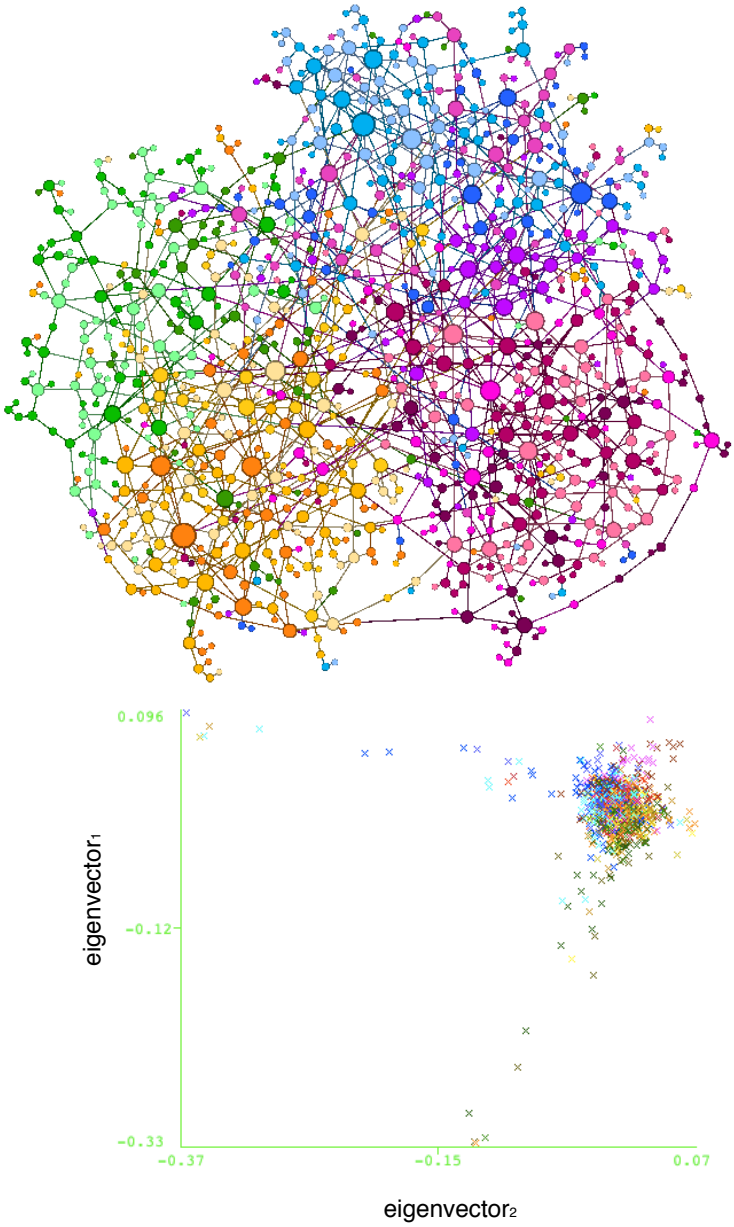


Figure 5.8: System with 1,000 agents, 16 roles, and one semantic service per agent. The network structure reflects the effects of integrating choice homophily in the system. Each node color reflects one role. Similar colors reflect that the roles are close to each other in the organizational ontology (see Figure 5.4). (Top) The network structure reflects the effects of introducing choice homophily into the system with the parameter $\varphi = 1$. (Bottom) Communities obtained through a clustering algorithm based on eigenvectors of the adjacency matrix [41].

color of the node represents the organizational role. Nodes with colors in the same range mean that their roles are close to each other in the organizational ontology (see Figure 5.4). Information about the structural properties of these networks such as average degree, path length, or clustering coefficient are described in Table 5.1. In this table, we have included the average choice homophily in the network $\overline{CH}$. This parameter measures the average homophily between an agent and its neighborhood . $\overline{CH}$ is calculated as follows:

$$\overline{CH} = \frac{\sum_{\forall a_i \in A} \sum_{\forall a_j \in N_i} \frac{CH(a_i, a_j)}{|N_i|}}{|A|} \tag{5.6}$$

The network shown at the top of Figure 5.7 is created considering only value homophily (φ = 0, see Definition 9). In this network, the agents are grouped based on similarity between services. The groups are tightly connected internally and there are few links that connect to other groups. Note that since organizational information has not been considered, in some communities, agents that offer similar semantic services but play different roles are connected. The communities obtained through a clustering algorithm are shown at the bottom of Figure 5.7. The clustering algorithm is based on the use of eigenvectors of the adjacency matrix [41]. The components of the first non-eigenvectors allow to detect communities. However, if there is a large number of group interconnections, the communities become more entangled and the community detection using this type of one-dimensional plot worsens. This difficulty can be solved by taking into account some more eigenvectors. In this case, we have considered two eigenvectors. The x-axis shows the components of the first non-trivial eigenvector. The y-axis shows the components of the second non-trivial eigenvector. The number of clusters obtained is 10. The clusters are clearly defined and loosely connected with other clusters. The network shown at the top of Figure 5.8 is created taking into account only status homophily (φ = 1). In this case, the consideration of information from a higher level of abstraction, such as organizational roles, facilitates the interaction among groups of agents that offer different services. At the bottom

of Figure 5.8, the clusters obtained are shown. Although the clusters in the figure cannot be easily distinguished, the algorithm detects 10 clusters again. The distances between different communities has been reduced considering organizational information, thereby facilitating the navigation among the communities. Nevertheless, when a query arrives at the appropriate community, it is more difficult to locate a service since no information related to services has been considered during the self-organization process. Section 5.6 discusses the effects of considering organizational information in the network creation process and in the service discovery process.

5.5 Decentralized Service Discovery Using Homophily

In open, large, and dynamic systems, agents should rely on local information during the service discovery process for several reasons. One reason is to prevent dependence on a single point of failure. Another reason is to avoid the effects of changes in the system structure. A third reason is that global information may not be available in open and dynamic systems. In this situation, it is important to provide agents of mechanisms that are based on local information.

In this section, we describe a service discovery process that relies on local information about the direct neighbors of the agents. Agents are able to locate the required service with only this information. The servide discovery strategy proposed exploits the structure based on homophily of the network to reduce the number of steps required to locate a suitable provider agent that offers the required service.

The selected algorithm for service discovery in the system is an extension of the Expected-Value Navigation (EVN) algorithm [128], which is a greedy, mixed algorithm that considers local information related to the similarity and the degree of connection. It has been modified to use choice homophily as the similarity measure that integrates organizational information with the service description. The proposed algorithm is called Choice Homophily Navigation (CHN). The CHN algorithm is based on a selection function that calculates the most promising neighbor a_j of an

agent a_i to reach the agent a_t is :

$$\mathcal{F}_{Ni}(a_t) = \underset{a_j \in N_i}{\operatorname{argmax}} P_s(\langle a_j, a_t \rangle) \tag{5.7}$$

For each neighbor a_j, $P_s(a_j, a_t)$ determines the probability that the neighbor a_j redirects the search to the nearest network community where there are more probabilities of finding the agent a_t .

$$P_s(\langle a_j, a_t \rangle) = 1 - \left(1 - \left(\frac{CH(a_j, a_t)}{\sum_{a_j \in N_i} CH(a_j, a_t)} \right) \right)^{|N_j|} \tag{5.8}$$

This probability uses homophily-based factors (choice homophily CH) and degree-based factors (number of neighbors $|N_j|$) to explore the network.

The CHN algorithm performs as follows (see Algorithm 2 and 3). When an agent a_i looks for an unknown target agent a_t (which provides a required service s_t and plays a certain role r_t), it sends a query to the most promising agent in its neighborhood. Likewise, when an agent a_i receives a query about a service that it cannot provide, it forwards the query to the most promising agent in its neighborhood. The most promising neighbor, $a_j \in N_i$, is the most similar neighbor to the target agent a_t and that has the highest number of connections. This process is repeated until an agent that offers a service that is 'similar enough' is found (Alg. 3 Line 7) or when the TTL (Time To Live) of the query ends (Alg. 3 Line 14). The criterion of 'similar enough' is established by the agent that generates the query as a semantic similarity threshold ε.

An example of this process is described considering the scenario presented in Section 5.2 (see Figure 5.1). Agent a_i should choose the most promising neighbor from its neighborhood (a_n,a_j, or a_k), to forward the query q. To do that, the agent a_i applies the function that appears in Equation 5.7. This function considers: (i) the

choice homophily between each neighbors of agent a_i and the profile of an unknown target agent $a_t = (rentalCar, Transport, \emptyset, \emptyset)$ that offers the service and plays the role specified in the query q; and (ii) the connection degree of the neighbors. Assuming the values of choice homophily that appear in Figure 5.1 ($CH(a_k, a_t) = CH(a_j, a_t) = 0.5$, and $CH(a_n, a_t) = 0.15$):

$$\begin{aligned}\mathcal{F}_{Ni}(a_t) &= \underset{a_k,a_j,a_n}{\text{argmax}} \left[1 - \left(1 - \frac{0.5}{1.15}\right)^5, 1 - \left(1 - \frac{0.5}{1.15}\right)^4, 1 - \left(1 - \frac{0.15}{1.15}\right)^5\right] \\ &= \underset{a_k,a_j,a_n}{\text{argmax}}[0.942, 0.897, 0.5] = a_k\end{aligned}$$

Therefore, agent a_i sends the query to the most promising agent, i.e agent a_k. This process is repeated until the similarity between a local services of an agent and the service in the query is over a threshold, or the query exceeds the TTL. In the described scenario, the process ends when the query arrives to agent a_v that is similar to the target agent a_t that a_i was looking for (see Figure 5.1b).

Algorithm 2 Function that describes how an agent a_i starts a service discovery process.

```
1: function startDiscovery()
2: TTL → 100                    /* value established by the agent a_i */
3: ε → 0.75                     /* value established by the agent a_i */
4: a_j ← serviceDiscovery(a_i, q = (a_i, s_q, r_q, TTL, ε))
5: if a_j ≠ ∅ then
6:     request(a_j, s_q)
7: end if
8: end function
```

5.6 Evaluation

In this section, we evaluate the proposed system structure and the service discovery strategy based on choice homophily. For the evaluation, we compare structural features and the success rate of the service discovery of our proposal with other network structures, commonly used in the complex networks area. The network structures that were considered in the experiments are:

Algorithm 3 Function that describes the service discovery process that an agent a_i carries out when receives a query.

```
1: function serviceDiscovery(a_i, q = (a_source, s_q, r_q, TTL, ε))
2: a_t ← (r_q, s_q, ∅, ∅)
3: if TTL > 0 then
4:     if CH(a_i, a_t) ≥ ε then
5:         inform(a_source, a_i)
6:     else
7:         a_i ← F_Ni(a_t)
8:         TTL ← TTL − 1
9:         serviceDiscovery(a_i, q = (a_source, s_q, r_q, TTL, ε))
10:    end if
11: else
12:    inform(a_source, ∅)
13: end if
14: end function
```

- *Random networks* (R), where links between agents are established randomly.

- *Scale-Free networks* (SF), where links between agents are established based on the degree of connection. Agents with a high degree of connection have a greater probability of receiving a new link than agents that have a low degree of connection.

- *Networks based on value homophily among agents* (Networks based on H_v). Agents in these networks are based on the value homophily and the degree of connection.

- *Networks based on choice homophily among agents* (Networks based on CH). Links in these networks are established based on the choice homophily between agents and the degree of connection.

We also compare our search strategy for decentralized service discovery with typical search strategies that are used in complex networks. The difference among them is how the most promising neighbor is selected in each step. These strategies are:

- *Random*: a search process that uses random walks [14, 51];

- *Degree*: a search process that uses only degree of connection information [144];
- *Similarity*: a search process that uses only service similarity information [146, 135, 11];
- *VHN*: a mixed search process that uses a combination of degree of connection and service similarity [128];
- *CHN*: a mixed search process that is based on the degree of connection and choice homophily $CH(a_i, a_j)$.

Queries in the experiments are uniformly generated among all the agents. This means that all the agents in the system had the same probability of generating service queries. A query consisted of two features that characterize the required provider agent: the semantic concepts that identified the organizational role and the semantic service description. Each query was forwarded by the agents following a criterion determined by the search strategy until the query was successfully solved or the query reached its TTL.

Each network of these experiments was undirected. Each agent played one role and offered one semantic web service. The agents were distributed uniformly over the roles. The role played by an agent was defined in a common organizational ontology (see Figure 5.4). All the experiments were performed with real semantic services. The set of semantic services used for the experiments were from the test collection OWL-S TC4 [1]. Each experiment has been done over 10 networks for each of the structures and 5,000 queries have been generated in each network.

The tests that we did for the evaluation of our proposal are divided in two groups: a first group of tests that evaluates configuration parameters related to networks based on CH and search strategies, and a second group of tests that compares our proposal

[1] http://www.semwebcentral.org/projects/owls-tc/

of network structure and search strategy based on CH with other existing structures and search strategies.

5.6.1 Evaluation of Configuration Parameters

In the following sections, we evaluate the effects that configuration parameters of networks based on CH have in the performance of service discovery. The parameters that we consider are related to the structure of networks based on CH (i.e., the influence of value and status homophily (φ), the size of the network, and the average number of neighbors), and the service discovery process (i.e., the similarity threshold (ε), and the Time To Live (TTL)). Specifically, we focus on how different values of these parameters affect the success and the average path length of the service discovery process.

5.6.1.1 Influence of Value and Status Homophily

The φ parameter balances the weight of value homophily and status homophily to determine the similarity measure between two agents (i.e., the importance of the role or service description in the overall similarity is regulated by φ)(see Equation 9). Details about the structural properties of these networks as φ changes are shown in Table 5.1. These properties are the following: $\overline{k}$ is the average degree of connection of the agents in the network; $\overline{c}$ is the average clustering coefficient [76] (indicates how nodes are embedded in their neighborhood); $\overline{d}$ is the average diameter of the network (the diameter is the longest graph distance between any two nodes in the network); $\overline{p}$ is the average distance between all pairs of nodes; $\overline{CH}$ is the average choice homophily in the network.

Figure 5.9 compares the results obtained with the different algorithms in networks based on CH varying the value of φ parameter. Each network has 1,000 agents and an average degree of connection of 2.5. In each graph, the φ parameter takes different

Properties / φ	$\overline{k}$	$\overline{c}$	$\overline{d}$	$\overline{l}$	$\overline{CH}$
0	2.5	0.0099	18.10	8.02	0.48
0.25	2.65	0.0062	16.20	7.05	0.42
0.5	2.70	0.0045	15.19	6.82	0.48
0.75	2.65	0.0052	15.2	6.84	0.58
1	2.67	0.0052	15.60	6.92	0.71

Table 5.1: Structural properties of *Networks based on CH* with different values of φ. $\overline{k}$ is the average degree of connection of the agents; $\overline{c}$ is the average clustering coefficient; $\overline{d}$ is the average diameter of the network; $\overline{p}$ is the average distance between all pairs of nodes; $\overline{CH}$ is the average choice homophily in the network

Al. / φ	Random	Degree	Similarity	VHN	CHN
0	39.22 $\pm$3.63	73.47 $\pm$9.96	45.25 $\pm$7.52	71.84 $\pm$13.31	71.96 $\pm$13.70
0.25	64.64 $\pm$2.18	89.19 $\pm$2.39	57.03 $\pm$3.08	85.51 $\pm$3.57	94.14 $\pm$0.93
0.5	65.85 $\pm$1.79	88.63 $\pm$1.96	63.15 $\pm$9.48	88.71 $\pm$3.27	95.55 $\pm$1.78
0.75	65.48 $\pm$2.74	88.85 $\pm$4.00	60.23 $\pm$5.94	87.03 $\pm$3.11	96.91 $\pm$0.85
1	64.64 $\pm$2.18	89.19 $\pm$2.39	95.14 $\pm$0.93	57.03 $\pm$3.07	85.51 $\pm$3.57

Table 5.2: Success rate (%) obtained by different search algorithms in *Networks based on CH* when φ value ranges from 0 to 1.

Al. / φ	Random	Degree	Similarity	VHN	CHN
0	31.72 $\pm$1.70	25.02 $\pm$5.84	19.36 $\pm$3.38	20.57 $\pm$2.88	20.52 $\pm$3.06
0.25	33.55 $\pm$1.04	21.63 $\pm$6.19	19.88 $\pm$1.87	16.99 $\pm$2.92	12.36 $\pm$0.98
0.5	33.44 $\pm$1.22	19.02 $\pm$4.77	19.48 $\pm$2.35	15.79 $\pm$1.86	11.38 $\pm$1.24
0.75	33.77 $\pm$1.07	18.78 $\pm$5.02	20.10 $\pm$1.63	16.83 $\pm$3.00	10.68 $\pm$1.76
1	33.66 $\pm$1.06	18.21 $\pm$2.96	21.83 $\pm$2.60	17.72 $\pm$2.32	12.08 $\pm$1.16

Table 5.3: Mean path length obtained by different search algorithms in *Networks based on CH* when φ value ranges from 0 to 1.

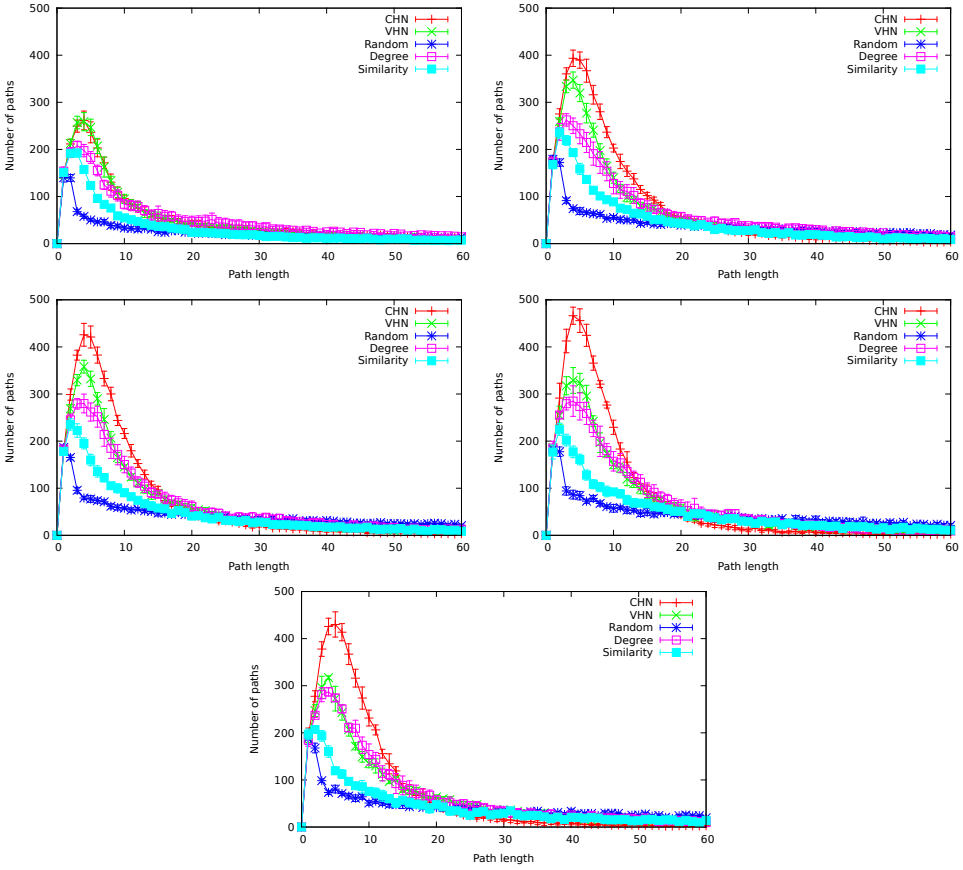


Figure 5.9: Search performance in *Networks based on CH* with different values for φ (Top row: φ [0, 0.25], Middle row: φ [0.5, 0.75], and Bottom row: φ=1).

values that range from 0 to 1 giving more importance to functional information (services) or to organizational information (roles). The x-axis shows the average number of steps required in the service discovery process. The y-axis shows the number of queries that were solved in a certain number of steps before the TTL (TTL=100, $\varepsilon = 0.75$). When $\varphi = 0$, the networks have been built using only value homophily information (see Figure 5.7). As φ increases, status homophily appears and organizational information has more influence in the network creation process (see Figure 5.8).

In general, it can be observed that, independently of the value of φ, the CHN algorithm obtains the best results providing with a greater number of short paths than other traditional algorithms used in distributed environments. The consideration of status homophily (i.e., role information $\varphi > 0$) improves the results obtained by all the search strategies but specially the results obtained by the CHN algorithm.

In networks that are built based only on semantic service information ($\varphi = 0$), several small agent communities that are specialized in certain types of services emerge and there are only a few connections between communities. In these communities, agents that play different roles but offer semantically similar services could be connected directly. These features make the navigation from one community to another more complicated. Consequently, the path lengths obtained by the search strategies are longer.

In the networks that are built based on a combination of organizational and service information ($\varphi = [0.5, 0.75]$), networks are not divided in loosely connected communities. Agents self-organize taking roles and services into account. Agents are connected to agents that play similar roles that are situated close to each other in the organizational ontology. Specifically, the best parameter configuration is when $\varphi = 0.75$. Furthermore, with low probability, agents also establish connection with agents that play completely dissimilar roles. This makes the navigation between communities easier. Consequently, the path lengths obtained in the search process are shorter. Organizational information is useful in guiding the search. Nonetheless, in networks that have been built with only this information, the search becomes complicated ($\varphi = 1$). The reason is that once an agent arrives to a community that has a high probability of containing the required service, it does not have any criteria for determining which agent is better for reaching the required service since all of them play similar roles.

In general, it can be observed that the consideration of both status and value homophily in the network structure improves the service discovery process. Also, the CHN algorithm significantly reduces the length of the paths to the target (see Figure

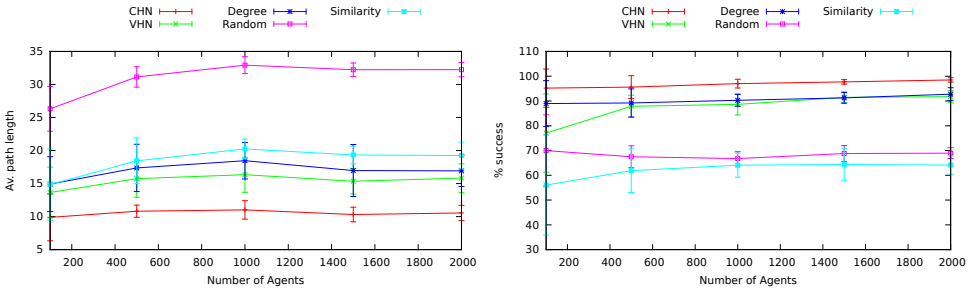


Figure 5.10: Influence of the number of agents on the average path length and success on *Networks based on CH*.

5.9). The success rate obtained by CHN increases when organizational information is included in the decision process, $\varphi > 0$ (see Table 5.2). The algorithm based on the degree of connection provides similar success rate; however, the mean path length is almost double the mean path length obtained with CHN algorithm.

5.6.1.2 Network Size

In order to check the scalability of network structures based on *CH*, we tested different search strategies when the number of agents changes and we paid attention to the average path length of successful searches and the percentage of success of the discovery processes. In this experiment, the average degree of connection of agents was 2.5, the TTL was 100, and the ε was 0.75. Figure 5.10 presents the results of the tests. The graphs show that the size of the networks does not have a significant influence in the path length and in the success rate when networks have more than 500 agents.

5.6.1.3 Neighbors

To evaluate the influence of the average degree of connection of agents in networks based on *CH* we considered values that range from 2 to 11. In this experiment,

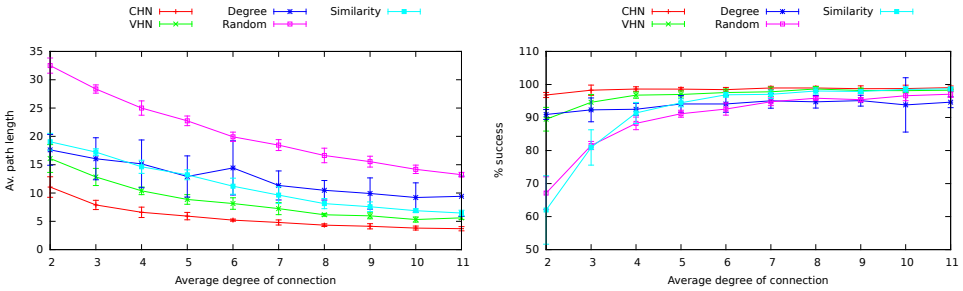


Figure 5.11: Influence of the average degree of connection on the average path length on *Networks based on CH*. The mean-shortest path length dismisses when agents have more connections.

networks had 1,000 agents, the TTL was 100, and the ε was 0.75. As it was expected, as the average degree of connection increased, the paths got shorter and it was easier to locate the target agent since agents had available more possibilities to guide the search (see Figure 5.11). Furthermore, the resulting networks were more robust. On the other hand, if agents joined one neighbor, the resulting structure was less robust and could be broken in isolated components under same failure conditions. In the generation process of networks based on *CH* we considered that agents when arrive to the system can establish a number of connections up to 2. This criterion creates networks with an average degree of 2.5 approximately. This degree of connection allows to analyze critical scenarios where the difference between search strategies and structures is significant.

5.6.1.4 Similarity Threshold

A query was successfully solved when an agent that offers a similar enough semantic service to the target service and played a similar role to the target role was found. This means that the semantic similarity between the services and organizational roles was over a threshold. We evaluated the effect of different ε values in the success rate and the average path length obtained with different search strategies in networks based on *CH* (see Figure 5.12). In this experiment, networks had 1,000 agents, each agent

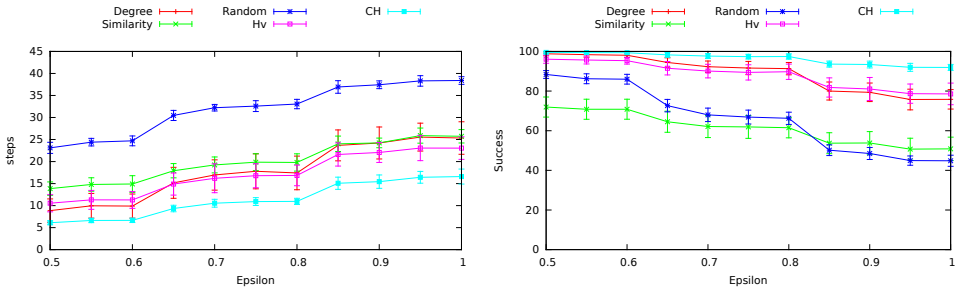


Figure 5.12: Influence on average path length and success of ε parameter on service discovery on *Networks based on CH* when different search strategies are used.

had an average degree of connection of 2.5, and the TTL was 100. We considered that 0.75 offers a balance between the degree of similarity and, the success rate and the average number of steps.

5.6.1.5 Time To Live

The Time To Live (TTL) parameter determines the maximum number of forwarding actions allowed in the system for a specific query. In this test, we evaluated how the variation of the value of this parameter affects to the discovery process in networks based on *CH*. In this experiment, networks had 1,000 agents, each agent had an average degree of connection of 2.5, and the ε was 0.75. The results are shown in Figure 5.13. As it was expected, higher values of TTL offered the opportunity to search strategies such as random walks and similarity to improve their success rate. The search strategy based on *CH* offered the shortest paths and the TTL had not a significant influence in its performance. In the rest of the experiments we considered a TTL=100 to give the chance to all the search strategies to reach the target.

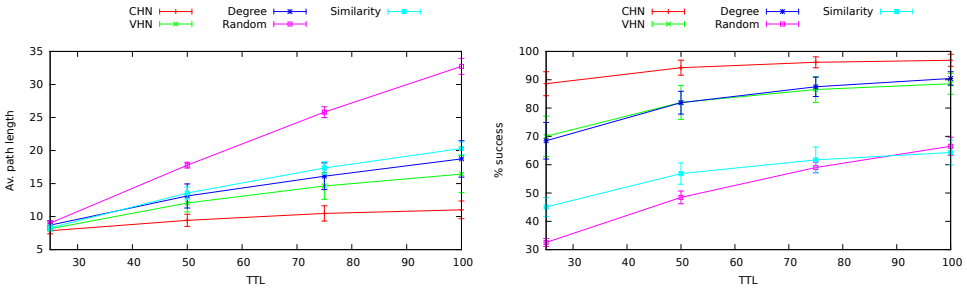


Figure 5.13: Influence on average path length and success of TTL parameter on *Networks based on CH* when different search strategies are used.

5.6.2 Service Discovery and Structure Evaluation

The aim of the second part of the tests is to compare our proposed structure and algorithm with other complex network models and search strategies. Specifically, we focus on two metrics: the average number of steps required by a search strategy to find the target agent and the percentage of successful searches obtained (searches that end before the TTL) in each network structure.

Based on the results obtained from the previous experiments, the setup for the second group of experiments was as follows. Each network of these experiments was undirected and had 1,000 agents. Each agent had an average degree of connection of 2.5. Agents played one role and offered one semantic web service. The agents were distributed uniformly over the roles. The role played by an agent was defined in a common organizational ontology (see Figure 5.4). All the experiments were performed with real semantic services. The set of semantic services used for the experiments were from the test collection OWL-S TC4 [2]. The value for the φ parameter for choice homophily between agents is 0.75. Each experiment has been done over 10 networks for each of the structures and 5,000 queries have been generated in each network. The value for the similarity threshold parameter ε is 0.75.

[2] http://www.semwebcentral.org/projects/owls-tc/

Properties / Network Structure	$\overline{k}$	$\overline{c}$	$\overline{d}$	$\overline{l}$	$\overline{CH}$
Random	2.53	0.0014	19.89	8.71	0.15
Scale-Free	2.48	0.0028	9.0	4.83	0.15
Net. based on H_v	2.53	0.0093	18.10	8.02	0.48
Net. based on CH	2.66	0.0058	15.80	6.91	0.57

Table 5.4: Structural properties of network structures.

5.6.2.1 Comparison with other Search Strategies and Complex Networks Models

We evaluated the performance of search strategies (random, degree, similarity, VHN, and CHN) in 10 networks for each type: networks based on H_v ($\varphi = 0$), networks based on CH (CH with $\varphi = 0.75$), Scale-Free (SF), and Random (R). Details about the structural properties of these networks such as average degree of connection, path length, or clustering coefficient are described in Table 5.4. In this table, we have included the average choice homophily in the network $\overline{CH}$. This parameter measures the homophily between an agent and its neighborhood.

With regard to the structural properties, SF and R networks are characterized by a low clustering. Moreover, the $\overline{CH}$ takes low values since choice homophily it is not considered in the network generation process. SF networks have a small diameter and short paths. Networks based on H_v and CH have low clustering values and high values of $\overline{CH}$. Networks based on CH have a higher value due to the consideration of organizational information. Note that the $\overline{CH}$ values are around 0.5. This is because the homophily criterion for establishing links do not limit agents to establishing links with only similar agents, but they can also establish links with dissimilar agents.

The results of this experiment are shown in Figures 5.14 and 5.15. The graph on the left contains four histograms (one for each network structure). In each histogram, the x-axis shows the different search strategies and the y-axis shows the average number of steps required to reach the target agent. In general, in each type of structure, the

shortest paths are obtained by the search strategies that are based on the criteria used to build the network. For example, random walks perform better in random networks or a degree-based method in SF networks. This highlights the close relationship between the search strategies and the network structures.

With regard to the average path length, the best structures that provide the shortest path are the networks based on CH and the SF networks. In the networks based on CH, the best strategy is the CHN. Moreover, strategies based on similarities, degree of connection, and the combination of both also provide short paths. In the SF networks, the best strategy is the strategy based on the degree of connection. Nevertheless, in structures of this type, strategies that combine degree of connection and similarity also obtain good results, particularly the CHN strategy. The conclusions to be drawn from this data are that Scale-Free networks and networks based on CH are structures where short paths can be found through different search strategies. Moreover, the CHN search strategy obtains good results in both structures: SF networks and networks based on CH.

In the graph on the right, the x-axis shows the different search strategies and the y-axis shows the percentage of successful searches (i.e, the percentage of queries that were solved before the TTL). For SF networks and networks based on CH the success in the service discovery process provide the best results with Degree, VHN, and CHN strategies. In SF networks, the success rate is over 80% with all the search strategies. In networks based on CH, the success rate is over the 80% in three search strategies. We can conclude that the search strategy that provides the highest success rate independently of the network structure is the CHN.

The path length and the success rate are closely related and can influence each other. For instance, even though short path lengths could be obtained in a network structure, this does not always mean that the structure offers good performance. The success rate could be too low, and only searches that are solved in the surroundings of the source agent (path with a low number of steps) are considered. For this reason, we have analyzed the relation between the success and the path length (PS) . This relation

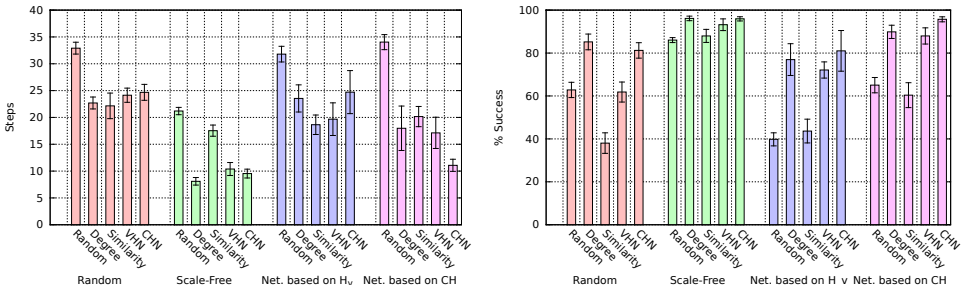


Figure 5.14: Search results using different network structures with an average connection degree of 2.5. (Left) Average number of steps in the discovery process. (Right) Success rate (%) of queries. The error interval is depicted with I at the top of each bar.

is measured using the following equation, which takes into consideration the number of queries that were successfully solved as well as the average mean path,

$$PS = \frac{\sum_{a_i \in A} \#sq(t)}{\#Q(t)} \cdot \frac{TTL - \bar{p}}{TTL} \tag{5.9}$$

In the equation, $\#sq(t)$ is the number of queries generated by an agent that were solved before the TTL at a given time t. The term $\#Q(t)$ is the total number of queries generated in the system at a given time t. TTL reflects the maximum path length allowed in the system, and $\bar{p}$ is the average path length (number of steps) of a service discovery process in the system. The first term evaluates the success rate and the second term evaluates the significance of the path length. The values of PS range in the interval [0,1], where 0 means that none of the queries generated in the system where solved, and 1 means that all the queries were successfully solved and by direct neighbors ($\bar{p} = 0$). This situation is possible in regular networks where each node is connected to the rest of the nodes of the network.

The results of PS obtained with the different network structures are shown in Figure 5.15. In this figure, the x-axis shows the different search strategies, and the y-axis shows the PS. In general, it can be observed that the success rate is the information

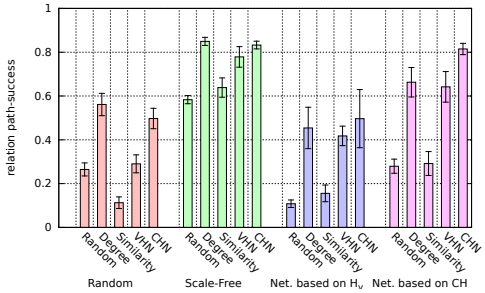


Figure 5.15: Relation between the path length and the success rate (PS) in different network structures with an average connection degree of 2.5.

that has the most influence on the PS. The PS values obtained confirm that SF networks and networks based on CH obtain the best results. The CHN is the strategy that offers better performance in different network structures than the other strategies that are commonly used in complex networks.

The results of this experiment allow us to conclude that degree-based algorithm and CHN perform well independently of the underlying network structure. Moreover, networks based on CH and Scale-Free networks have desirable characteristics for providing an underlying structure to a discovery system. They have high percentage of success in the search process and short paths. Therefore, the traffic generated by the service discovery process is reduced and its efficiency improved.

5.6.2.2 Tolerance to Failures

Networks could be sensitive to failure or deliberate attacks. The most critical situation for networks where the distribution of the degree of connection follows an exponential or power-law distribution is when deliberate attacks on highly connected nodes are produced. In the context of service discovery, the failure of an agent implies the removal of all its links. We evaluate the performance of the service discovery process in different network structures as the number of failed nodes increases. We

have studied the failure tolerance of networks based on *CH* and SF networks in different situations, but in this article, we only include a subset of these experiments. Specifically, we focus on 'sabotage' situations in two particular network structures, networks based on *CH* and SF networks. We have only included these two types of networks since both of them obtained the best results in the experiments described above. We only show the results obtained in scenarios where deliberate failure ('sabotage') is produced. We consider this scenario to be a more interesting scenario than random failures since the network structures and the service discovery strategies are evaluated in the worst case.

Figure 5.16 shows the behavior of networks based on *CH* and Scale-Free networks under 'sabotage'. The results for the Scale-Free networks are shown in the left column. The results for the networks based on *CH* are shown in the right column. The top row shows the average number of steps required to reach the target agent as the number of failures of highly connected agents increases (50,100,150, and 200). Taking into account the average path length of the successful searches, the results indicate that the SF networks obtain shorter paths than networks based on *CH* in the presence of failures. When the number of failures is 50, this difference is not as significant. This difference is greater when the number of agents that have failed is between 50 and 100. The main reason of this fact is that SF networks are more sensitive to failures; therefore, as the number of highly connected nodes that fail increases, the network is divided in a higher number of isolated parts where the only successful searches are those which can be solved by a nearby agent.

The middle row shows the success rate in the discovery process as the number of agents that fail increases. The graph on the right shows that the rate of success in networks based on *CH* is over 40% until 150 agents are removed (using the CHN search strategy). In the case of SF networks (left), the success rate is seriously reduced when more than 50 agents fail (5% success when the number of deleted agents is 150). Note that, in the case of SF networks, the tests range from 50 to 150 failure agents. This is because in the experiments with 200 failure agents, the network is

disconnected in so many isolated parts that it is not possible to find the required services. The PS relation is shown in the bottom row. The network structure based on *CH* offers the best results where path length and success rate under intentional failures is concerned. Moreover, the strategy based on *CH* also obtains the best results in networks based on homophily and almost the same as the strategy based on the degree of connection in SF networks.

In general, it can be observed from the results that since SF networks have a distribution of the degree of connection that follows a power-law, SF networks are more vulnerable to intentional failures. This is due to the existence of hubs that concentrate network connections. If an attack is addressed to these hubs, the network can be broken into isolated groups. However, a growing network based on choice homophily generates structures with an exponential degree distribution. In networks based on choice homophily, the size of the hubs is limited. For example, whereas a SF network with 1,000 agents and an average degree of connection of 2.5 can contain nodes connected to more than 75 nodes, a network with an exponential distribution degree of connection barely arrives to hubs with 20 connections with other nodes. The absence of highly connected hubs makes the network more robust under a deliberate attack.

Based on these results, we consider that the most suitable structure for the self-organization of services is the network structure based on choice homophily between agents. When agents join other similar agents, the network has short paths to locate the desired services. Moreover, a hill climbing mechanism can be implemented with a high success rate in the service discovery process. Therefore, despite the network structure has not small-world network properties, its performance is in the same order of magnitude.

Finally, the proposed CHN algorithm, which uses node degree and the homophily information, performs as the best method in SF networks and *CH* based networks. Networks based on *CH* are more robust than SF networks under targeted attacks. Therefore, homophily-based networks seem a good structure for self-organized sys-

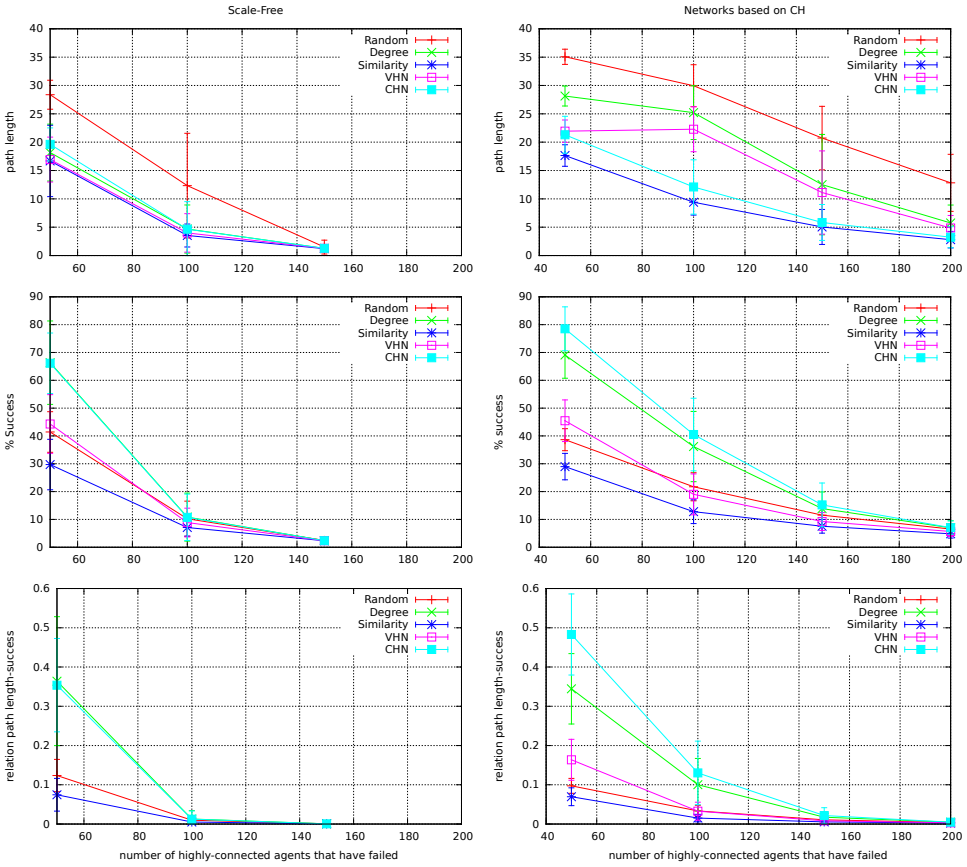


Figure 5.16: Sabotage in Scale-Free networks (Left column) and in *Networks based on CH* (Right column). (Top row) The average number of steps required to reach the target using different search strategies as the number of agents that have failed increases. (Middle row) The success rate (%) in the service discovery process using different search strategies as the number of agents that have failed increases. (Bottom row) The relation between path length and success (PS) using different search strategies as the number of agents that have failed increases.

tems and the CHN algorithm offers a good performance even in other type of networks that are not self-organized considering homophily criterion.

5.7 Conclusions

In this chapter, we have studied how the integration of different areas such as Service-Oriented systems, MAS, Semantics, and Complex Networks provide the necessary tools to build a decentralized service management system. We have proposed a Service-Oriented MAS where agents establish relations with other agents by taking into account a social feature that is present in complex networks and that acts as a self-organizing criterion. This feature is called homophily. Specifically, we have considered a type of homophily called *choice homophily* that is composed of the combination of *value* and *status homophily*. In the context of agents, choice homophily is based on the attributes of agents. *Value homophily* is based on the services provided by the agent. *Status homophily* is based on the organizational role played by the agent. Choice homophily is used to create a network based on preferences without the supervision of a central authority where agents have a greater probability of establishing links with other agents that share attributes with them (such as services and roles) than with dissimilar agents. Therefore, the system does not need an initial training period to establish its structure. The resultant structure is a growing network based on preferences. The degree of connection of this type of networks follows an exponential distribution.

Moreover, in the presented model, agents only have to maintain their local view and do not have to store information about routes that could frequently change. The proposed algorithm for decentralized service discovery is based on semantic information and considers local information about choice homophily between agents in its decision process.

Several experiments have been performed to evaluate and compare our network model (network based on CH) and service discovery strategy (CHN) with other existing proposals in Complex Networks. We evaluated the influence of the inclusion of organizational information in networks based on CH and in the search strategy CHN. The consideration of the organizational information in the network structure and in

the service discovery process considerably improves the performance of the system providing a high success rate and short paths. Furthermore, the proposed model to build networks based on homophily creates network structures with a degree of connection that follows an exponential distribution. This means that there are no hubs as in SF networks that follow a power-law distribution. Therefore, the system is more robust under critical situations such as sabotage. In these scenarios, their performance is better than other network structures such as SF networks.

We also compared our proposal with other network structures and algorithms. The results indicate that the service discovery strategy allows agents to locate the required service in just a few steps, not only in structures that are built following homophily criteria but also in other networks such as SF networks. The performance of the algorithm CHN in networks based on CH and SF networks is very similar considering the average path length of the searches in the discovery process and the success rate. Although networks based on CH do not have hub nodes such as SF networks that facilitate the search process reducing the number of steps, agents considering the homophily information are also able to provide short paths similar to those obtained by SF structures.

CHAPTER

6 Self-Organization Mechanisms in Service-Oriented MAS

6.1 Introduction

The evolution of Internet and communications, and the emergence of new business models have generated new requirements in the underlying systems such as the dynamic adaptation to changes in the environment in order to improve the system performance. In distributed large-scale systems, where there is a lack of global knowledge, dynamic adaptation generates new challenges [1] [13]. In this context, the inclusion of self-organization mechanisms plays an important role to facilitate the evolution of these systems. Researchers have focused on the integration of self-organization functionalities in distributed systems. *Self-organization* is considered a bottom up mechanism or process that enables a system to arrange its organiza-

tion at run-time, without explicit external commands [39]. Starting from entities that are structured in a sub-optimal organization or that are not organized at all, a self-organizing system is able to form a specific organization to pursue a well-defined goal [75]. The main issue in self-organization is to determine the best mechanism for reorganizing the current structure through the execution of local actions in order to achieve the desirable behavior despite a high degree of uncertainty in the system. Self-organization mechanisms attempt to deal with this task. The inclusion of these mechanisms in distributed systems provides desirable system features such as openness, robustness, flexibility, or scalability [143]. However, the main goal is the improvement of the system utility in dynamic environments. In order to facilitate the integration of self-organization mechanisms, it is desirable for the systems to have three main features: (i) no external control, central authority, or supervisor should guide the adaptation process; (ii) the system should be able to evolve; (iii) the agents of the system should be able to deal with uncertainty in order to take decisions.

Service discovery systems are deployed in dynamic environments where their components, features, and tasks do not remain constant. These systems are expected to perform well under many circumstances (i.e., when the number of available agents changes, or when the service demand varies with time). However, the majority of the proposals for service discovery in distributed systems are only focused on the location task and do not take into consideration the inclusion of self-organization mechanisms in order to adapt their structure to environmental conditions and requirement changes [105]. The structural adaptation plays an important role in service discovery since structural relations determine the interactions between the agents, their local knowledge, and, therefore, the performance of the service discovery process. Moreover, since there is not a global view of the system in large, open, and distributed systems, this adaptation should be performed in a decentralized way without the supervision of any centralized authority and considering only local knowledge. This ensures that the system is robust under failures.

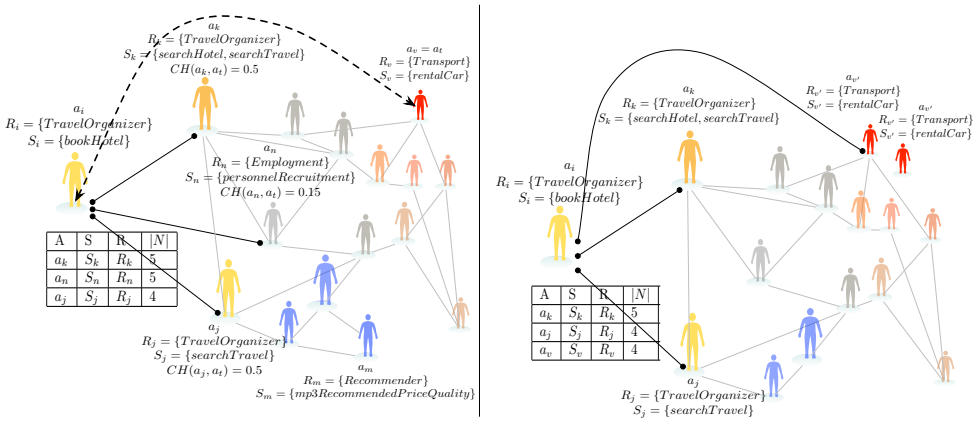
In this chapter, we present two self-organization mechanisms that are included in

the service discovery process in order to facilitate system adaptation when changes in service demand occur. One mechanism focuses on how the relations between agents could be rearranged to improve system performance. The other mechanism considers the adaptation of the agent population according to the service demand. The main advantages of this proposal are that the self-organization of the system is a continuous process that is carried out by each individual agent without central supervision; each agent is able to reason about when it is most appropriate to make a self-organization decision; agents only require local information about the service demand and the utility of their links; and, system dynamics about structural relations and population are taken into account.

The rest of the chapter is structured as follows: In section 6.2, we describe an example of a service discovery scenario. In section 6.3 the information that agents considers to make decisions is described. Section 6.4 describes the reasoning process that agents follow to decide about self-organization actions. In section 6.5, we present a set of experiments to validate the proposed model and the self-organization mechanisms in different scenarios. Finally, in section 6.6, conclusions and final remarks are presented.

6.2 Self-Organized Service Discovery Scenario

To illustrate the context where the self-organization mechanisms are applied, let us to consider the service discovery scenario presented in Section 5.2. Basically, the scenario consists on a set of agents that play organizational roles and offer semantic web services. The structural relations between these agents have been established taking the choice homophily criterion into account. Agents, in some situations, should interact with each other to achieve a task that they cannot afford to do individually since they are not specialized in that area or because the task is too complex to be carried out by a single agent. Moreover, we assume that the service demand changes at different times of day. Therefore, agents should be able to evaluate their importance in



(a) . **(b)** .

Figure 6.1: Example of a decentralized service discovery system where agents have self-organization mechanisms.

the system and adapt their structural relations with other agents to deal with changes in environment conditions trying to optimize the overall performance of the system.

In the scenario shown in Figure 6.1a, agent a_i should choose one of its neighbors, a_n,a_j, or a_k, to forward the query q. In order to select the most promising neighbor, the agent i applies Function 5.8. This function considers: (i) the choice homophily between the neighbors of a_i and a *fictitious* agent $a_t = (rentalCar, Transport, \emptyset, \emptyset)$ that offers the service and plays the role specified in the query q; and (ii) the degree of connection of the neighbors. Assuming the values of choice homophily that appear in Figure 6.1 ($CH(a_k, a_t) = CH(a_j, a_t) = 0.5$, and $CH(a_n, a_t) = 0.15$) and the degree of connection of the agents, agent a_i sends the query to the most promising agent (i.e., agent a_k). This process is repeated until the similarity between a local service of an agent and the service in the query is over a certain threshold or the query exceeds the TTL. In the described scenario, the process ends when the query arrives to agent a_v (see Figure 6.1a). Afterwards, agent a_i stores agent a_v in its local view as a possible candidate for establishing a future structural relation if some of its current relations are not being used currently.

We assume that as time passes service demand changes and the services offered by agents that play the role $Transport$ start to be the most demanded services in the system. As a result, agent a_i analyzes its internal state st_i and realizes that its structural relation with neighbor a_n has not being used but it has knowledge about an agent from a previous interaction that offers services that now are being demanded. Therefore, agent a_i decides to break its current structural relation with a_n and establishes a new one with the candidate that was discovered as a result of a previous search process ($(a_i, a_n) \rightarrow (a_i, a_v)$) (see Figure 6.1b). This self-organization action reduces the path distance towards the agents that provide the most demanded services in a certain moment and improves the success rate in future discovery processes.

Also, there are other agents (such as agent a_m) that, through an analysis of the information in their internal state, realize that they are offering services that are not being demanded in the system. In this case, these agents might decide to leave the system. For instance, in Figure 6.1b, agent a_m leaves the system. Otherwise, if the demand for services offered by agents that play certain roles increases, agents might decide to create a clone to satisfy the current service demand. Therefore, the population of the system self-adapts to the demand in each moment. In our scenario, assuming that the demand for services offered by agents with role $Transport$ increases considerably, agent a_v creates a clone, $a_{v'}$, to satisfy the current service demand.

In our model we include two adaptation mechanisms that allow agents to reason about different organization actions based on their local view. These organization actions allow the adaptation of the structural links between agents and the system population when service demand changes.

6.3 Agent Internal State

Our self-organization model allows agents to reason about actions that they can carry out in order to improve the service discovery activity in the system. Specifically, this model allows agents to analyze their local information and, based on this information,

to determine whether it is appropriate to replace their structural relations that are not being used with profitable ones or to leave, continue, or clone themselves in order to adapt the system population to the service demand.

Each agent in the system maintains a local view of what is happening. This local view is stored in its *internal state*, st_i, (see Definition 2) which is built using the information collected from previous interactions with other agents. The st_i allows an agent to make decisions based on its local data without the need for communicating with other agents, which requires coordination with higher level entities that supervise what is happening in the system and introduces scalability problems [13]. The st_i of an agent in the system is defined by the following tuple $< \mathcal{K}_i^N \cup \mathcal{K}_i^A, \mathcal{K}_i^E, \mathcal{K}_i^{st} >$ where:

- $\mathcal{K}_i^N \cup \mathcal{K}_i^A$ represents the *partial knowledge about its neighborhood.* An agent only has knowledge about a limited number of agents and only has a partial view of them. Specifically, an agent has knowledge about the following:
  - a set of direct neighbors $\mathcal{K}_i^N =< \{R_j, |N_j|, UQ_{ij}\} \forall j \in N_i >$. Agent a_i contains information about each neighbor j: the roles j plays, the degree of connection of j, and the number of times that a query that arrived to the agent a_i was not forwarded through its neighbor a_j;
  - a set of acquaintances $\mathcal{K}_i^A = \{a_k \in A | a_k \notin N_i\}$ that agent a_i is at least aware of them as a result of the discovery process.

  The neighborhood of an agent does not remain constant. The update process of neighbors is carried out by the agents in a proactive way as a consequence of the service discovery activity in the system: new agents are discovered; other agents decide to leave since they are not receiving enough requests related to their services; or existing links are reinforced or replaced depending on whether or not they are being used in the forwarding process.

- $\mathcal{K}_i^E$ is *the model of the local environment*. This model includes information that estimates aspects of the environment that are relevant to the agent in order to

improve the agent's situation in the system. In the case of our service discovery scenario, an agent maintains the following information $\mathcal{K}_i^E =< \vec{q_i}, Q_i, \rho_i, >$, where:

- $\vec{q_i} = [q_i^{r_1}, q_i^{r_2}, ...,]$ is the local service demand distribution (i.e., the number of queries that the agent receives about services offered by different organizational roles);
- Q_i is the number of total queries that the agent receives and forwards;
- ρ_i is the correlation coefficient that establishes the relationship between the local service demand distribution $\vec{q_i}$ and an estimation of the expected service demand distribution. The correlation parameter ρ_i ranges in the interval [-1,1], where 1 indicates a perfect positive fit and -1 indicates a perfect negative fit. If there is no linear correlation, ρ_i is close to 0.

The information contained in the model of the local environment is important for determining what the most demanded services are. Moreover, this information is continuously updated by the continuous interactions among agents and helps agents to learn about remote parts of the system.

- $\mathcal{K}_i^{st}$ is the *status* of the agent. The status depends on the significance of the information an agent has. If an agent has an accurate view of the system, it is considered to be in a stable situation. When a new agent arrives to the system, or when it has outdated information that introduces noise in its local environment, the agent is considered to be in a transition situation. It is important to determine the situation of an agent in order to make decisions related to the adaptation process. Agents in the system can be in one of two adaptation status: *transition* and *stable*. An agent is in a *transition* state when its local view of the service demand in the system ($\mathcal{K}_i^E$) does not follow the expected service distribution ($eDist$). All the agents are initially in a *transition* status since agents do not have information about the service demand distribution in the system. An agent is in a *stable* status when its local view of the service demand follows the expected service demand distribution.

Power-law, Exponential, and Zipf's-law distributions are present in many features of Internet [3, 61]. In our system, the exponential distribution has been considered as the function that models the service demand in the system, where there are always a few services that are the most demanded and the rest of the services have a lower demand rate. Specifically, we assume that the expected service demand distribution is $eDistr(x) = a \cdot e^{x \cdot b}$, where the x parameter represents a role identifier. We estimate the a and b parameters of this distribution using the least squares method and the data from $\vec{q_i}$.

An example of $\mathcal{K}_i^E$ is shown in Figure 6.2a. The graph reflects the local view of the service demand distribution $\vec{q_i}$ in a certain moment. The x-axis shows the numeric identifiers for the roles $(r_1, r_2, ...)$ that appear in the queries that agent a_i receives. The y-axis shows the number of queries forwarded of each role $\vec{q_i}[r]$. The number of queries is normalized. The line shows an estimation of the expected distribution of service demand taking into account the data collected by the agent. We consider that this distribution follows an exponential function since Power-law, Exponential, and Zipf's-law distributions are present in many features of Internet [3, 61]. In our system, the exponential distribution has been considered as the function that models the service demand in the system, where there are always a few services that are the most demanded and the rest of the services have a lower demand rate. Specifically, we assume that the expected service demand distribution is $eDistr(x) = a \cdot e^{x \cdot b}$, where the x parameter represents a role identifier. We estimate the a and b parameters of this distribution using the least squares method and the data from $\vec{q_i}$. Specifically, in Figure 6.2a, the a and b parameters take the following values: $a = 1.73$ and $b = 1.16$. The correlation coefficient $\rho_i = 0.86$ near 1 indicates that the current data about service demand $\vec{q_i}$ fits with the expected exponential distribution $eDistr$.

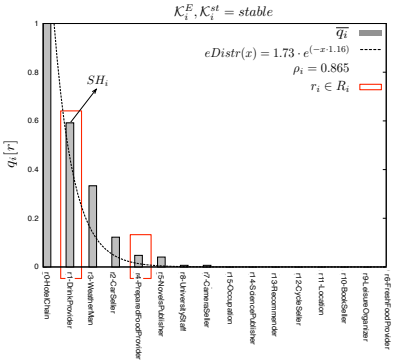
Figure 6.2 also shows how the internal state st_i of an agent is modified as the service demand changes. Specifically, in Figure 6.2a, agent a_i has its status $\mathcal{K}_i^{st} = stable$. However, when the service demand changes in the system, the agent should be able to notice this change. In Figure 6.2b, agent a_i is able to recognize that there are cer-

tain services that are starting to be demanded. These services are offered by agents that play roles r_{15}, r_{14}, r_{13}, and $r_1 1$. The correlation ρ_i between the local data and the expected service demand distribution decreases considerably (initially the ρ_i was 0.865 and now it is 0.59). The evolution of ρ_i indicates that agent a_i is aware that there is a change in the service demand distribution. Therefore, previous information is not meaningful for analyzing the current situation, and agent a_i decides to reset its local model of the environment ($\mathcal{K}_i^E$). Figure 6.2c shows how agent a_i has modified its local service demand distribution $\vec{q_i}$ towards a new service demand distribution where services offered by agents that play roles r_{15}, r_{14}, and r_{13} are the most demanded. Note that agent a_i has changed the order of the role identifiers based on the demand for their services. At that moment, agent a_i does not have an accurate view of the service demand since ρ_i is 0.54. Figure 6.2d shows how agent a_i has continued reorganizing its local service demand distribution. Now, the correlation parameter ρ_i between the data and the estimated service demand distribution has increased ($\rho_i = 0.89$); therefore, agent a_i considers that its local data accurately reflects the current service demand. As a result, agent a_i could consider that it is an appropriate moment to execute possible self-organization actions.

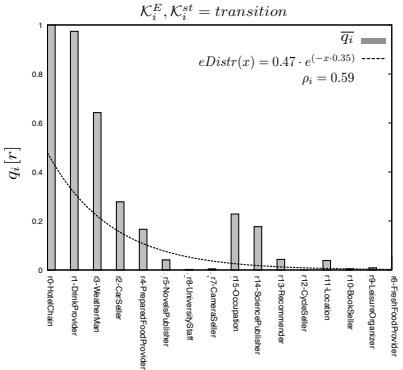
6.4 Agent Reasoning Process

Agents reason about local properties of the system by combining their *partial view of the neighborhood* ($\mathcal{K}_i^N \cup \mathcal{K}_i^A$), their *model of the local environment* ($\mathcal{K}_i^E$), and their *internal status* ($\mathcal{K}_i^{st}$). As a result of this reasoning process, agents make local decisions that are translated into actions. These actions affect the internal state of the agents, st_i, which should be updated. The environment is also affected by the local actions carried out by the agents, since the structural relations and the population are adapted to changes in service demand. These local actions improve the global performance of the system.

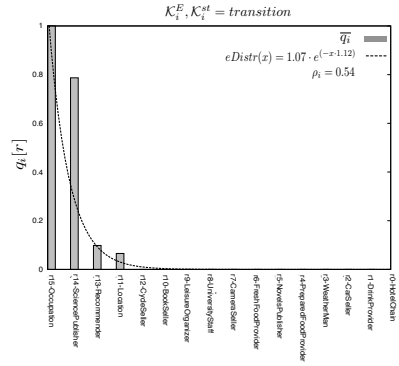
Algorithms 4 and 5 shows an overview of the reasoning process followed by the



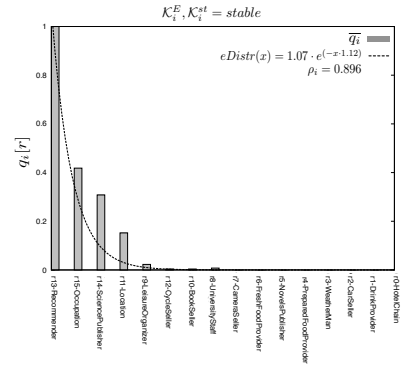
(a) Initial st_i adapted to the current service demand. The agent fixes the distribution of roles. ρ_i is near 1 and $\mathcal{K}_i^{st}$ is *stable*.



(b) st_i when service demand changes. ρ_i is not near 1 and $\mathcal{K}_i^{st}$ is *transition*.



(c) st_i during the adaptation to new service demand. The agent reorder the distribution of roles. ρ_i is not near 1 and $\mathcal{K}_i^{st}$ is *transition*.



(d) st_i after the adaptation to new service demand. The agent fixes the distribution of roles. ρ_i is near 1 and $\mathcal{K}_i^{st}$ is *stable*.

Figure 6.2: Example of the internal state st_i of the agent a_i where the $\mathcal{K}_i^E$ and $\mathcal{K}_i^{st}$ are shown.

agents during the discovery process where self-organization mechanisms are considered. These algorithms are modifications of the Algorithms 2 and 3 in Chapter 5. The modifications are lines 7-14 in Algorithm 4 and line 8 in Algorithm 5. This process is initiated when agent generates a query $q = (a_i, s_q, r_q, TTL, \varepsilon)$, which contains the identifier of the agent that sends the query, the semantic description of the desired service, the role that the target agent should play, the Time To Live, and the similarity

threshold ε (see Alg. 4).

The agent that receives the query creates the profile of a *fictitious* agent a_t with the service and role specified in the query q (Alg. 5 Line 2). Agent a_i looks for a neighbor similar to a_t. If it finds a suitable neighbor, the service discovery process ends (Alg. 5 Lines 4-6) and informs to the agent that generates the query. Otherwise, the agent a_i forwards q to one of its neighbors $a_j \in N_i$. Specifically, q is forwarded to the agent that has semantic closeness (degree of choice homophily) to the *fictitious* agent a_t and also has a high degree of connection (see Equation 5.7)(Alg. 5 Line 8). Then, agent a_i updates its information about: which of its links have been used, the number of total queries it received (Q_i), and the number of queries about the role r_q ($\vec{q_i}[r_q]$).

Finally, if the query is solved successfully, the source agent (which started the service discovery process) sends a request to the provider agent that was found. If the source agent does not already have another agent in its acquaintances that plays the role contained in the query, the source agent adds the provider agent to its set of acquaintances (Alg. 4 Lines 8-10). Finally, the source agent updates its internal state st_i and analyzes the set of self-organization actions that it can carry out (Alg. 4 Lines 12-14).

Algorithm 4 Function that describes how an agent a_i starts a service discovery process where agents consider self-organization actions.

1: **function** $startDiscovery()$
2: $TTL \rightarrow 100$ /* number of steps associated to the query */
3: $\varepsilon \rightarrow 0.75$
4: $a_j \leftarrow serviceDiscovery(a_i, q = (a_i, s_q, r_q, TTL, \varepsilon))$
5: **if** $a_j \neq \emptyset$ **then**
6: $request(a_j, s_q)$
7: /* START MODIFICATION */
8: **if** $\nexists a_k \in \mathcal{K}_i^A : r_q \in R_k$ **then**
9: $\mathcal{K}_i^A \leftarrow \mathcal{K}_i^A \cup a_j$
10: **end if**
11: **end if**
12: $InternalStateAnalysis(a_i)$
13: $LinkDecayAdaptation(a_i)$
14: $PopulationAdaptation(a_i)$ /* END MODIFICATION */
15: **end function**

In the following sections, we are going to describe the main functions that appear in

Algorithm 5 Function that describes the service discovery process that an agent a_i carries out when receives a query and considers self-organization actions.

function $serviceDiscovery(a_i, q = (a_{source}, s_q, r_q, TTL, \varepsilon))$
$a_t \leftarrow (r_q, s_q, \emptyset, \emptyset)$
if $TTL > 0$ **then**
  if $CH(a_i, a_t) \geq \varepsilon$ **then**
    $found \leftarrow true$
    $inform(a_{source}, a_i)$
  else
    $a_i \leftarrow promisingNeighbor(a_i, a_t)$ /* MODIFICATION */
    $TTL \leftarrow TTL - 1$
    $serviceDiscovery(a_i, q = (a_{source}, s_q, r_q, TTL, \varepsilon))$
  end if
else
  $inform(a_{source}, \emptyset)$
end if
end function

Algorithm 6 Function that describes how an agent a_i selects the most promising neighbor when considers self-organization actions.

function $promisingNeighbor(a_i, a_t)$
$a_n \leftarrow \mathcal{F}_{N\,i}(a_t)$
$updateLinksDecay(a_i, a_n)$
$\vec{q_i}[r_q] \leftarrow \vec{q_i}[r_q] + 1$
$Q_i \leftarrow Q_i + 1$
$a_i \leftarrow a_n$
return a_i
end function

the service discovery algorithm: *Internal state analysis*, *Structural adaptation*, and *Agent adaptation*.

6.4.1 Internal State Analysis

The internal state analysis consists of reorganizing the local information that an agent has about the service demand and determining whether or not this information is reliable and sufficient to be able to know what is happening in the system (see Algorithm 7).

Initially, if the agent is in a *transition* state, the agent can reorganize the local service demand distribution $\vec{q_i}$ taking into account the number of queries received about the

services of each role (Lines 2-4). An example of this situation is shown in Figures 6.2c and 6.2d where agent a_i is in a *transition* state and decides to reorganize the local service demand distribution $\vec{q_i}$ ($\{q^{r_{15}} \gg q^{r_{14}} \gg q^{r_{13}} \gg q^{r_{11}} \gg q^{r_{12}} \gg q^{r_{10}} \gg q^{r_9} \gg q^{r_8}\}$ to $\{ q^{r_{13}} \gg q^{r_{15}} \gg q^{r_{14}} \gg q^{r_{11}} \gg q^{r_9} \gg q^{r_{12}}\}$). If the agent is in a *stable* state, the previously defined order of most demanded services is maintained.

Once $\mathcal{K}_i^E$ is updated, the agent updates its *status* $\mathcal{K}_i^{st}$ (Lines 5-9). To determine if it is in the appropriate status, the agent evaluates the linear *correlation*. Correlation parameter ρ_i indicates the degree of fitness between the local data $\vec{q_i}$ and the expected exponential distribution $eDistr(x) = a \cdot e^{x \cdot b}$. If ρ_i is over a certain threshold δ, the local information accurately reflects the current traffic situation and the agent changes its current status to *stable*.

The consideration of an outdated $\mathcal{K}_i^E$ could negatively influence the reasoning process of the agent. This usually happens when frequent dynamic changes in the service demand occur. In order to determine whether or not reset its model of the local environment $\mathcal{K}_i^E$ and its status $\mathcal{K}_i^{st}$ to *transition*, the agent should receive a sufficient number of queries. The significance of the number of queries received is evaluated through a logistic function (Lines 10-24) :

$$P(Q_i) = \frac{1}{1 + \cdot e^{\frac{-(Q_i - d)}{y}}}, \tag{6.1}$$

where y is the slope, d is the displacement constant, and Q_i is the number of queries the agent has forwarded. The most influential constant is d. A higher value of d means that the agent is going to consider a higher number of queries in order to make a decision about resetting the information in $\mathcal{K}_i^E$. The function $P(Q_i)$ returns a value in the range [0,1], where 0 indicates that the agent has not received a sufficient number of queries to make a decision about resetting its current $\mathcal{K}_i^E$ and where 1 indicates that the number of queries is significant enough to make a decision. Besides the number of queries, one of the following cases must also be given in order to reset

$\mathcal{K}_i^E$:

- the agent is in a *stable* status, but its ρ_i is under a certain threshold (Line 12);
- the agent should be in a *stable* status since it has received a high number of queries, but it is still in a *transition* status (Line 16).
- the agent is in a *stable* status, but it has never consider to clone itself (Line 20).

All three of these situations mean that the local model of the agent starts to be outdated with respect to the current system demand, and it is advisable to reset its local view since the consideration of outdated information introduces noise in the current data distribution and affects the self-organization process.

Algorithm 7 Function that analyzes the internal state of the agent: reorganizes the local model of the environment, changes the status of the agent, or resets its current information when it considers that is required.

1: **function** $InternalStateAnalysis(i)$
2: **if** $(\mathcal{K}_i^{st} = TRANSITION)$ **then**
3: $\quad sort(\mathcal{K}_i^E)$
4: **end if**
5: $a, b \leftarrow leastSquaresFitting(\vec{q_i})$
6: $\rho_i \leftarrow linearCorrelation(a \cdot e^{(b \cdot x)}, \vec{q_i})$
7: **if** $(\rho_i > \delta)$ **then**
8: $\quad \mathcal{K}_i^{st} \leftarrow STABLE$
9: **end if**
10: $pReset \leftarrow 1/(1 + \cdot e^{-(Q_i - d)/y})$
11: **if** $(pReset > random)$ **then**
12: $\quad$ **if** $((\mathcal{K}_i^{st} = STABLE) \wedge (\rho_i < \delta)$ **then**
13: $\quad\quad \mathcal{K}_i^E \leftarrow resetInformation(\mathcal{K}_i^E)$
14: $\quad\quad \mathcal{K}_i^{st} \leftarrow TRANSITION$
15: $\quad$ **end if**
16: $\quad$ **if** $(\mathcal{K}_i^{st} = TRANSITION)$ **then**
17: $\quad\quad \mathcal{K}_i^E \leftarrow resetInformation(\mathcal{K}_i^E)$
18: $\quad\quad \mathcal{K}_i^{st} \leftarrow TRANSITION$
19: $\quad$ **end if**
20: $\quad$ **if** $((\mathcal{K}_i^{st} = STABLE) \wedge (clones = 0))$ **then**
21: $\quad\quad \mathcal{K}_i^E \leftarrow resetInformation(\mathcal{K}_i^E)$
22: $\quad\quad \mathcal{K}_i^{st} \leftarrow TRANSITION$
23: $\quad$ **end if**
24: **end if**
25: **end function**

Once the agent has analyzed its internal state, it is able to make decisions about

changing its structural links or about remaining, cloning, or leaving the system. In the following sections, we explain the information that an agent takes into account in order to make each decision and the reasoning process that the agent follows.

6.4.2 Self-Organization of the Structural Links

Agents are able to reason about whether or not to maintain, reinforce or create new structural relations. To facilitate the reasoning process about the structural relations between agents, agents consider a *decay* metric that it is associated to each link. This metric indicates the probability of maintaining a link. It ranges in the interval [0,1], where 0 indicates that the link is not being used and 1 indicates that the link is being used. The function is a sigmoid :

$$decay(UQ_{ij}) = 1 - \frac{1}{1 + \cdot e^{\frac{-(UQ_{ij}-z)}{y}}}, \tag{6.2}$$

where y is the slope and z is the displacement constant. The constant that has more influence on the decay function is z. In Figure 6.3, the effects of varying this constant in the function can be observed. If z takes a high value, the agent is more resilient to making changes in its current links. UQ_{ij} is the number of queries that arrived to agent a_i and were not forwarded through agent a_j. Each time agent a_i forwards a query, it updates the information about the traffic of its links. If the query is forwarded through agent a_j, the UQ_{ij} is updated to 0. Otherwise, the UQ_{ij} is increased by increments of 1 (see Algorithm 6, Line 3). With the information provided by the *decay* function, agent a_i reasons about the benefit of maintaining its current links.

In Algorithm 8, we describe the reasoning process that agents follow to adapt their current links. Each agent invokes this function when a service discovery process that the agent has initiated ends. If the agent has at least one acquaintance ($|\mathcal{K}_i^A| > 0$) in its internal state st_i as a result of a previous discovery process, then the agent analyzes its current structural links with its neighbors (Line 2). The agent has information

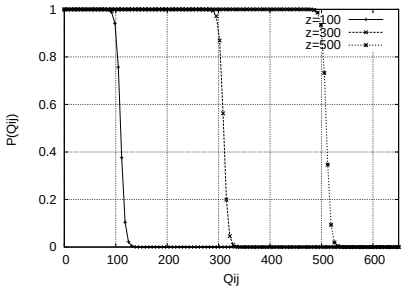


Figure 6.3: Decay function for the structural links of agents with different values for the displacement parameter z.

about its neighborhood $\mathcal{K}_i^N$ in its st_i. With this information, the agent evaluates the probability of maintaining each of its links (Line 4). If this probability is under a certain threshold, the agent looks for a candidate in $\mathcal{K}_i^A$. The agent selects the acquaintance that plays one of the most demanded roles according to its local view of the environment $\mathcal{K}_i^E$ (Lines 6-10). In the case that a suitable candidate is found by the agent, the agent breaks its current relation and establishes a new one with the selected acquaintance. Finally, the agent updates its internal state (Lines 11-15).

6.4.3 Population Self-Organization: Leave, Clone, or Remain

The analysis that evaluates whether it is worthwhile for the agent to remain in the system, clone itself, or leave the system takes the following three parameters into account:

- the number of queries received by the agent Q_i (see Equation 6.1),
- the degree of correlation ρ_i,
- the structural homophily of the agent SH_i.

Algorithm 8 Function that analyzes the traffic through each link in order to decide if it is appropriate to modify them. Each agent invokes this function when a service discovery process that it initiated ends.

```
1:  function LinkDecayAdaptation(a_i)
2:  if |K_i^A| > 0 then
3:      for a_j ∈ K_i^N do
4:          decay ← decay(UQ_ij)
5:          if decay < random then
6:              while ¬found ∧ a_n ∈ K_i^A do
7:                  if r_n ∈ demandedRoles then
8:                      found ← true
9:                  end if
10:             end while
11:             if found then
12:                 K_i^A ← K_i^A − a_n
13:                 K_i^N ← K_i^N ∪ a_n
14:                 K_i^N ← K_i^N − a_j
15:             end if
16:         end if
17:     end for
18: end if
19: end function
```

In the context of service discovery, we define the concept of *structural homophily* SH_i as the degree of similarity between the services demanded in the system and the services provided by an agent in the system. This kind of homophily reflects how important an agent is to the system with regard to the current service demand. Structural homophily is used to facilitate the decentralized self-organization of the system population. In the system, each agent controls the queries that it receives. The agent classifies each query taking into account the organizational role associated to its $\vec{q_i}$. The agent stores this information and periodically analyzes its structural homophily, (i.e., the agent determines how similar the services it offers are to the services required in the system).

DEFINITION **10** *(Structural homophily) The structural homophily of an agent with respect the system dynamics is defined by the following function :*

$$SH_i = a \cdot e^{r_i \cdot b} \tag{6.3}$$

where r_i is the role of agent a_i that maximizes the following function:

$$r_i = \underset{x \in R_i}{\operatorname{argmax}}\, a \cdot e^{x \cdot b} \tag{6.4}$$

where the a and b parameters are obtained in the InternalStateAnalysis function (see Algorithm 7 Line 5). SH_i ranges in the interval [0,1], where 1 indicates that the services the agent offers are required in the system, and 0 indicates that the services the agent offers are not being demanded in the system.

An example of how the *structural homophily* of agent a_i is calculated is shown in Figure 6.2a. Agent a_i plays two roles: r_1 and r_4. At that moment, using the data in $\vec{q_i}$, the exponential function that estimates the service demand distribution is $eDistr = 1.73 \cdot e^{-x \cdot 1.16}$. Therefore, the structural homophily of agent a_i is:

$$SH_i = 1.73 \cdot e^{-r_1 \cdot 1.16} = 0.54, \tag{6.5}$$

where

$$r_1 = \underset{r_1, r_4}{\operatorname{argmax}}[1.73 \cdot e^{-r_1 \cdot 1.16}, 1.73 \cdot e^{-r_4 \cdot 1.16}] = \underset{r_1, r_4}{\operatorname{argmax}}[0.54, 0.018] \tag{6.6}$$

This means that the services that agent a_i offers are being demanded, but are not the most demanded services in the system.

The population adaptation algorithm (see Algorithm 9) evaluates whether or not it is worthwhile for an agent to remain in the system. Basically determines one of three possible actions: *remain* in the system, *leave* the system, or create a *clone*.

The analysis of the *leave* action is based on the following parameters: *number of queries received* Q_i, the *degree of correlation* ρ_i, and SH_i (Lines 2-8). If the number of queries received is high enough, ρ_i is over a certain threshold, and SH_i has a value near 0; then the agent decides to leave the system. However, this does not always happen. In order to ensure the availability of a certain type of services in

the system, the agent does not leave the system if there is no similar neighbor that provides similar services (Line 4). Finally, if the agent leaves the system, it breaks all the connections with all its immediate neighbors and communicates that it is going to leave. The neighbors will try to find an alternative neighbor based on the choice homophily connection criterion.

If the agent has decided not to leave the system, it analyzes the *clone* action (Lines 9-14). This analysis is also based on the three parameters described above. The main difference is the logistic function for evaluating the significance of the *number of queries received*. In this function, the displacement parameter d takes into account the number of clones that an agent has:

$$P(Q_i, clones) = \frac{1}{1 + \cdot e^{\frac{-(Q_i - 2^{clones})}{y}}}, \tag{6.7}$$

where the parameter y is the slope, Q_i is the number of queries the agent has received, and 2^{clones} is the displacement. A higher value of the displacement implies that an agent must receive more queries in order to consider a clone action. The higher the number of clones that an agent creates, the higher the number of queries should receive; therefore, the next time the probability of cloning decreases exponentially. In our proposal, it is assumed that there are unlimited resources. Therefore, the displacement depends on the number of clones that an agent has created previously. However, in scenarios where there are a limited economical or physical resources, the displacement could be expressed in terms of other variables.

If the number of queries received is high enough, ρ_i is over a threshold, and SH_i has a value near 1; then the agent decides to execute the *clone* action (Line 10). However, this does not always happen. In order to prevent the number of clones increasing exponentially, there are two more conditions that reduce the probability of cloning. The agent does not clone if all its neighbors are similar to it or if the number of queries it forwards has not increased since the last analysis (Line 11). Taking into account all these parameters, the agent evaluates whether or not creating a clone is

worthwhile. The clone generated by the agent will offer the same services and play the same roles, and the number of clones it has will be initialized with the value of its father. The cloned agent establishes links with other agents in the system taking into account the choice homophily criterion. When an agent creates a clone it resets its internal state.

Algorithm 9 Function that decides the most appropriate action taking into account the current local view of the agent: remain in the system, leave the system, or clone itself. Each agent invokes this function when a service discovery process it initiated ends.

function PopulationAdaptation(a_i)
$pLeave \leftarrow 1/(1 + \cdot e^{-(Q_i - d')/y})$
if $((\mathcal{K}_i^{st} = STABLE) \wedge (SH_i < random) \wedge (pLeave > random))$ **then**
  if (similarN(N_i) >0) **then**
    leave()
    leave ← true
  end if
end if
$pClone \leftarrow 1/(1 + \cdot e^{-(Q_i - 2^{clones})/y})$
if $(\mathcal{K}_i^{st} = STABLE) \wedge (\rho_i > \delta) \wedge \neg leave$
$\wedge(SH_i) > random))$ **then**
  if $((similarN(N_i) <\mid N_i \mid) \wedge (\Delta q^i > 0))$ **then**
    clone()
  end if
end if
end function

6.5 Evaluation

Several tests were performed to evaluate the effects of the introduction of adaptation mechanisms in a decentralized service management system. There were three sets of tests. Each one analyzes the effects of a different adaptation mechanism on the system performance. The first test set analyzes the influence of changing structural relations between agents. The second test set is focused on the effects of agent decisions about remaining in the system, leaving the system, or cloning themselves. The third test set that pays attention to the benefits of integrating both the changes in structural relations and agent decisions about continuing, leaving, or cloning.

Specifically, the tests focus on a set of metrics that are meaningful for the analysis

of the performance of the system and for the effects on the service discovery process when agents incorporate self-organization mechanisms [66] [114]. These metrics are:

- Average number of steps required to locate an appropriate agent that solves a query.
- Percentage of queries that are solved before the TTL.
- Communication load improvement: This measures the system improvement comparing the number of exchanged messages during the service discovery process when adaptation mechanisms are exploited with respect to the number of exchanged messages when the system is not self-organized.

$$CL = 1 - \frac{\text{number of messages generated in a self-organized system}}{\text{number of messages generated in a system that is not self-organized}} \tag{6.8}$$

- Progress: This refers to how the system progressively improves its performance using a self-organization mechanism.
- Time for adaptation or Latency: This is the time needed to recover the normal behavior of the system after a change.
- Structural adaptive cost: This quantifies the number of structural changes required to adapt the system:
  - Number of structural relations between agents that have changed during the service discovery process,
  - Number of agents that clone or leave the system during the service discovery process.

Each set of tests has a set of 5 networks (undirected preferential attachment networks) with 1,000 agents. Agents play one role and offer one semantic web service associated to this role. Initially, agents are uniformly distributed over 16 roles, which are

defined in an organizational ontology. The set of semantic service descriptions used for the experiments have been taken from the OWL-S TC4 test collection [1].

All the agents in the system have the same probability of generating service queries. A query consists of two features that characterize the required provider agent: the role and the service. The query is successfully solved when an agent that offers a similar service (i.e., the degree of semantic match between the semantic service descriptions is over a threshold $\varepsilon = 0.75$) is found before the TTL ($TTL = 100$). We assume that all agents are collaborative, that is, agents will fulfill the rules and redirect the queries.

Query distribution in the system is modeled as an exponential distribution ($\lambda = 0.7$) where there are services offered by certain roles that are the most demanded and the rest of services have a lower demand rate [3, 61]. In the experiments, we made a snapshot of all the metrics every 10,000 queries in order to see the evolution of the system.

6.5.1 Changing Structural Relations

The first set of tests evaluated the effects of introducing self-organization mechanisms that modify the structural links between agents. Specifically, we evaluated the mechanism based on the decay of the structural links between agents. We compared the obtained results with another mechanism that is based on a Reinforcement Learning (RL) algorithm called Weighted Policy Learner (WPL)[2]. We chose this algorithm to compare our approach since Reinforcement Learning is a common approach for solving multi-agent decision problems and specifically, the proposal presented in [2] is the first one to study and analyze the interaction between learning and self-organization.

WPL uses a learning strategy that is similar to WoLF [18]. As in many RL algorithms, WPL makes use of two matrices, π_i and Q_i, for each agent. In the the context of

[1] http://www.semwebcentral.org/projects/owls-tc/

service discovery, π_i maintains the probability of sending a query about a service that should be provided by an agent that play the role r through the structural relation with a neighbor j. Thus, the organizational roles are the states of π_i and the actions are the neighbors. The matrix Q_i stores the rewards, which are based on the success of the previous searches. π_i values are initialized using semantic similarity values. The WPL algorithm is based on the following idea: to slow down learning when moving away from a stable policy and to speed up learning when moving towards the stable policy. The decision-making algorithm for establishing when it is appropriate to add or remove a link is based on a re-organization parameter (P_o), and on the average degree of connection of the network.

In the experiments, we considered different values for configuration parameters that are meaningful in both self-organization mechanisms. Specifically, we analyzed the influence of parameters that make agents be more resilient to structural changes or be more prone to making structural changes. In the case of our self-organization mechanism, which is called `Decay-based`, this parameter is the displacement z (see Equation 6.2). In the case of the reinforcement learning mechanism, which is called `RL-based`, this parameter is the re-organization parameter P_o. We evaluated both mechanisms and the influence of their configuration parameters in two different scenarios. In the first scenario we tested several values for the configuration parameters z and P_o. We performed an analysis to determine the configurations that offered the best results for each mechanism. In the second scenario, we only considered these configurations under changes in the service demand distribution.

6.5.1.1 First Scenario: service demand does not change.

In the first scenario, the agents were initially distributed over the different organizational roles uniformly, and the queries that agents generated followed an exponential distribution ($\lambda = 0.7$) over the organizational roles. We chose 4 for the average degree of connection of each agent because with `RL-based` mechanism a lower value divided the network into isolated parts. By considering local information generated

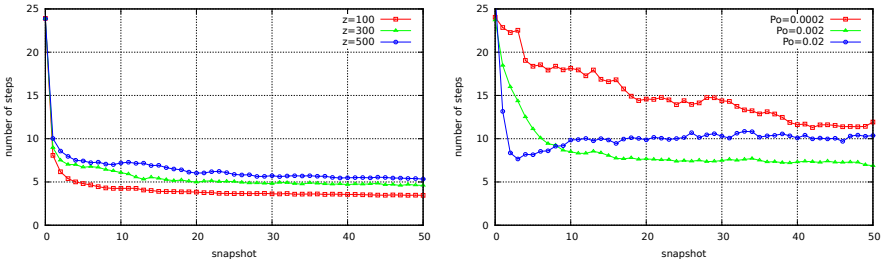


Figure 6.4: Average path length obtained when agents use link-based organization mechanisms: (Left) `Decay-based`, (Right) `RL-based`.

during the discovery process, agents should be able to reorganize their links according to the service demand.

Each graph has an associated table that contains the results with error intervals obtained in certain snapshots sn. The results obtained in the first snapshots ($sn = 1$ and $sn = 5$) are shown since there were more significant differences between mechanisms and configurations. The last snapshot $sn = 50$ is also shown since it reflects the final results when the system was adapted. For reasons of clarity, the error intervals of the results are not shown here in the graphs.

Figure 6.4 shows that the introduction of self-organization mechanisms considerably improves the number of steps required to reach a suitable provider agent. The x-axis shows the snapshots, and the y-axis shows the average number of steps required to reach a provider agent that solves the query. Initially, the average number of steps was near 25 and decreased to 7 steps in a few snapshots. In the case of the `RL-based` mechanism, the re-organization parameter with value $P_o = 0.002$ offered better results than with the other values (see Table 6.1 (Left)). In these configurations, agents are prone to change their structural links. The `Decay-based` is less sensitive to configuration parameters than `RL-based`, and, in general, the `Decay-based` improvement in the average path length was better than the improvement obtained with `RL-based` mechanism.

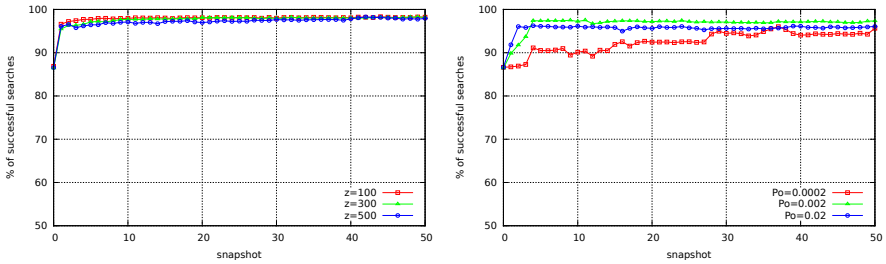


Figure 6.5: Percentage of successful searches (searches that are solved before the TTL) when agents use link-based self-organization mechanisms: (Left) `Decay-based`, (Right) `RL-based`.

	Av. path		
	$sn = 1$	$sn = 5$	$sn = 50$
$\mathbf{z = 100}$	8.07 ± 1.03	4.81 ± 0.51	$\mathbf{3.45 \pm 0.24}$
$\mathbf{z = 300}$	8.96 ± 0.78	6.70 ± 0.76	$\mathbf{4.59 \pm 0.52}$
$z = 500$	10.02 ± 2.35	7.42 ± 0.80	5.33 ± 0.98
$P_o = 0.0002$	22.85 ± 5.63	18.37 ±7.45	23.76 ± 6.90
$\mathbf{P_o = 0.002}$	18.46 ± 2.34	11.12 ±6.25	**6.87 ± 0.53**
$P_o = 0.02$	13.16 ±6.41	8.14 ± 1.14	10.36 ± 2.25

	% Success		
	$sn = 1$	$sn = 5$	$sn = 50$
$\mathbf{z = 100}$	$\mathbf{96.59 \pm 0.94}$	97.71 ± 0.89	$\mathbf{98.21 \pm 0.23}$
$z = 300$	95.47 ± 1.72	97.12 ± 1.5	$\mathbf{98.18 \pm 0.54}$
$z = 500$	96.09 ± 1.42	96.42 ± 1.06	97.94 ± 0.92
$P_o = 0.0002$	86.73 ± 18.70	90.52 ± 15.06	95.65 ± 3.78
$P_o = 0.002$	89.83 ± 15.14	97.35 ± 0.82	$\mathbf{97.34 \pm 1.20}$
$\mathbf{P_o = 0.02}$	91.83 ± 7.32	96.10 ± 5.11	96.07 ± 3.15

Table 6.1: (Left) Average path length, and (Right) percentage of successful queries in different snapshots sn when agents change their structural relations using `Decay-based` or `RL-based` mechanisms in the discovery process.

Figure 6.5 shows the effects on the success of the service discovery process. The x-axis shows the snapshots, and the y-axis shows the percentage of queries that were solved by a provider agent before the TTL. In general, both approaches improved the percentage of queries that ended successfully in the system. This improvement was achieved in the first snapshots and then remained constant. In the case of the `Decay-based` mechanism, there was no difference between the results obtained with the different values of the z parameter. In the case of the `RL-based` mechanism, the difference between the results obtained with $P_o = 0.002$, $P_o = 0.02$ and $P_o = 0.0002$ was more relevant in the first adaptation steps, and then this difference decreased. It can be concluded that systems that require a faster adaptation should use values between 0.002 and 0.02 for parameter P_o.

Figure 6.6 shows the number of structural relations that agents change in order to improve the system performance. The results show that the `Decay-based` mechanism allowed agents to be aware that there was a change in the service demand; therefore, they realized that structural changes were needed to adapt some of their links according to a new service demand. This fact can be observed in the first five snapshots where the number of rewired links is greater than in the following snapshots. If the rewiring action implies a cost, the most suitable configuration is $z = 300$ since agents consider the rewiring action, but the number of structural changes is not as significant as with the configuration $z = 100$. In the case of the `RL-based` mechanism, this only occurred when the re-organization parameter was $P_o = 0.02$ or $P_o = 0.002$. With lower values of P_o, the agents were less prone to make many structural changes and the structural changes followed a constant rate.

Figure 6.7 shows the improvement in the communication load when the system is adapted with respect to when agents do not include adaptation mechanisms. In general, it can be observed that both mechanisms considerably reduced the communication load. The `Decay-based` mechanism introduced an improvement of over 0.5 independently of the values of the parameter z. In the case of the `RL-based` mechanism, the communication load is also decreased, but it took more time and there

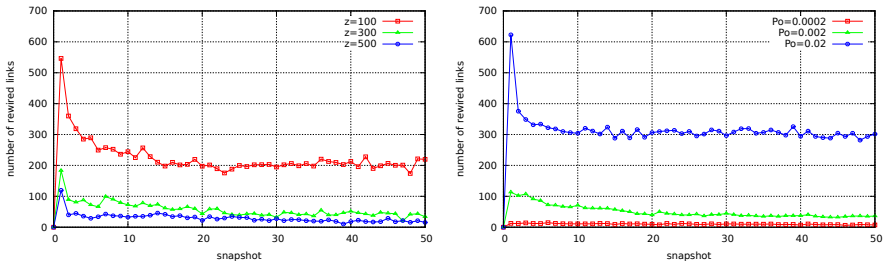


Figure 6.6: Number of structural relations rewired when agents use link-based organization mechanisms: (Left) `Decay-based`, (Right) `RL-based`.

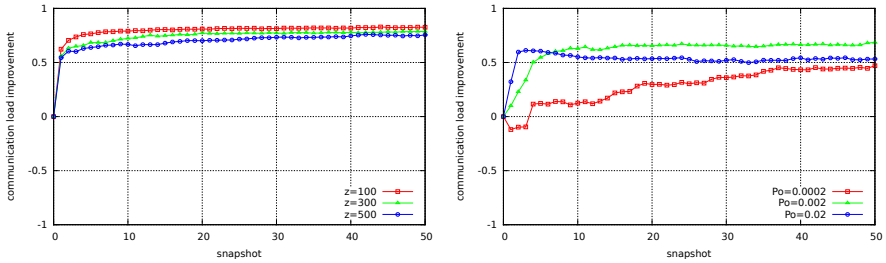


Figure 6.7: Communication load metric when agents use link-based self-organization mechanisms: (Left) `Decay-based`, (Right) `RL-based`.

	Number of rewired links		
	$sn = 1$	$sn = 5$	$sn = 50$
$z = 100$	546.2 ± 37.44	289.4 ± 62.38	219.8 ± 78.25
z = 300	**183.4 ± 41.73**	**72.2 ± 20.68**	**34.0 ± 14.45**
$z = 500$	119.6 ± 23.79	28.8 ± 12.72	15.6 ± 14.69
$P_o = 0.02$	12.6 ± 3.63	12.0 ± 5.11	7.4 ± 2.65
P_o = 0.002	**113.4 ± 28.17**	**86.0 ± 4.11**	**36.2 ± 8.16**
$P_o = 0.02$	622.6 ± 91.84	333.6 ± 35.59	301.4 ± 47.40

	Communication load improvement		
	$sn = 1$	$sn = 5$	$sn = 50$
z = 100	0.62 ± 0.02	0.76 ± 0.03	**0.82 ± 0.01**
$z = 300$	0.55 ± 0.03	0.68 ± 0.05	0.78 ± 0.02
$z = 500$	0.54 ± 0.05	0.63 ± 0.04	0.75 ± 0.01
$P_o = 0.0002$	-0.12 ± 0.41	0.12 ± 0.51	0.47 ± 0.27
P_o = 0.002	0.09 ± 0.45	0.54 ± 0.20	**0.68 ± 0.039**
$P_o = 0.02$	0.32 ± 0.39	0.60 ± 0.17	0.53 ± 0.10

Table 6.2: (Left) Number of structural relations rewired, and (Right) Communication load improvement in different snapshots sn when agents use link-based self-organization mechanisms: `Decay-based`, and `RL-based`.

was more variability in the results. This is because the adaptation process with this mechanism takes more time.

In general, the `Decay-based` mechanism provides a more reactive adaptation behavior that makes agents change many structural relations when they detect a significant change in its internal state. The best configurations were obtained with $z = 100$ and $z = 300$. These two configurations were similar. Both offered good results in path length, success, and communication load. The best results are obtained by the configuration with $z = 100$. However, this configuration made a high number of structural changes. In scenarios where the cost of the structural changes is significant, this configuration may not be appropriate, and the configuration with $z = 300$ could be considered to be more suitable in reduce the costs. The `RL-based` mechanism with parameter configurations $P_o = 0.002$ and $P_o = 0.02$ considered more structural changes than $P_o = 0.0002$. Therefore, the behavior of the `RL-based` mechanism with $P_o = 0.002$ and $P_o = 0.02$ is more appropriate for dynamic environments whereas $P_o = 0.0002$ is suitable for less dynamic environments where the service demand distribution remains without changes during a long period of time after a change.

6.5.1.2 Second Scenario: dynamic service demand.

In the second scenario, the link-based self-organization mechanisms were evaluated taking into account a service demand distribution that changes over time. The service demand distribution changed in intervals of 500,000 queries. During these intervals, snapshots were made every 10,000 queries in order to see the system evolution. There were two intervals. In the first interval (snapshots in the range [0,50]), the service demand followed an exponential distribution ($\lambda = 0.7$). In the second interval (snapshots in the range [50,100]), the service demand also followed an exponential distribution. However, in the second interval, the most demanded services were those that were less demanded in the previous interval. In this second scenario, we only show the results obtained with the best configurations of each self-organization mechanism: for `Decay-based` mechanism $z = 300$ and $z = 100$, and for `RL-based` mechanism $P_o = 0.002$.

Figure 6.8 (Left) shows the effects of adaptation mechanisms on the path length under dynamic service demand. For the `Decay-based` algorithm, the adaptation mechanism considerably reduced the average number of steps required to reach the target service in the first interval. There was no a significant difference between the results obtained with `Decay-based` and different values of z. The adaptation achieved in the first interval was better than in the following interval since in the first interval there was no historical information about different previous service demands, which introduces noise in the internal state of the agents. The agents should realize that they must reset their current view of the service demand, and afterwards, collect information about the new service demand in order to accurately analyze the utility of their links. In snapshot 50, there was a sharp rise, which indicates that the service demand changed and the system was not adapted. At the beginning of the second interval, the service demand changed to the worst case, which was the inverse service demand distribution. For this reason, in the second interval, the number of structural changes required was greater than in the previous interval, and the differences between the results obtained with `Decay-based` and different values for z parameter were more significant. The best results were obtained with $z = 100$, which was the configuration for agents that were more prone to make structural changes. Note that in this second interval, the error intervals were bigger (most of all at the beginning of the second interval), which means that there was more variability in the number of steps required to find a service (see Table 6.3). Something similar happened with the `RL-based` adaptation mechanism. However, there were a few differences with respect to the `Decay-based` mechanism. The average path length obtained using the `RL-based` mechanism was longer (7 steps in the first interval and 10 steps in the second interval). Moreover, the error intervals were larger, which introduces more uncertainty in the results (see Table 6.3).

Figure 6.8 (Right) shows the success rate in different intervals. Both adaptation strategies obtained good results (over the 95% in the majority snapshots). This percentage decreased sharply at the beginning of the second interval, when there was a considerably large change in the service demand distribution. However, the system

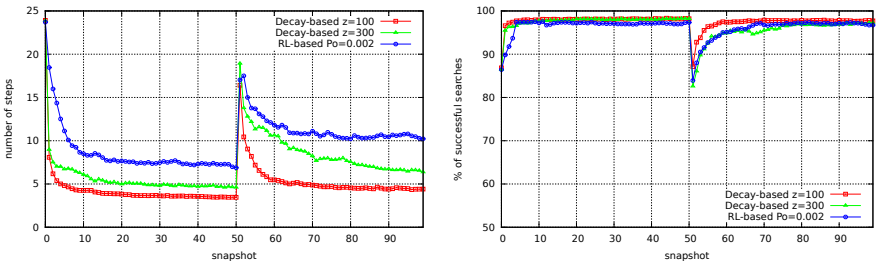


Figure 6.8: (Left) Average number of steps required to reach the desirable provider agent, and (Right) percentage of successful queries when there are dynamic changes in the service demand and agents use link-based self-organization mechanisms: `Decay-based` and `RL-based`.

	Av. path					
	$sn = 1$	$sn = 5$	$sn = 50$	$sn = 51$	$sn = 55$	$sn = 99$
$\mathbf{z = 100}$	8.07 ± 1.03	4.81 ± 0.51	3.45 ± 0.24	16.40 ± 2.16	7.18 ± 0.54	$\mathbf{4.41 \pm 0.32}$
$z = 300$	8.96 ± 0.78	6.70 ± 0.76	4.59 ± 0.52	18.92 ± 3.89	11.32 ± 1.30	6.37 ± 1.11
$P_o = 0.002$	18.46 ± 2.34	11.12 ± 6.25	6.87 ± 0.53	17.02 ± 5.98	13.69 ± 0.92	10.21 ± 1.36

	% Success					
	$sn = 1$	$sn = 5$	$sn = 50$	$sn = 51$	$sn = 55$	$sn = 99$
$\mathbf{z = 100}$	96.59 ± 0.94	97.71 ± 0.89	98.21 ± 0.23	87.14 ± 2.99	96.38 ± 1.47	$\mathbf{97.70 \pm 0.59}$
$\mathbf{z = 300}$	95.47 ± 1.72	97.12 ± 1.5	98.18 ± 0.54	82.57 ± 15.63	92.51 ± 5.08	$\mathbf{97.19 \pm 2.0}$
$P_o = 0.002$	89.83 ± 15.14	97.35 ± 0.82	97.34 ± 1.20	83.92 ± 4.67	92.69 ± 8.39	96.72 ± 2.32

Table 6.3: (Up) Average path length, and (Down) Percentage of successful searches in different snapshots sn when service demand changes and agents use link-based self-organization mechanisms: `Decay-based`, and `RL-based`.

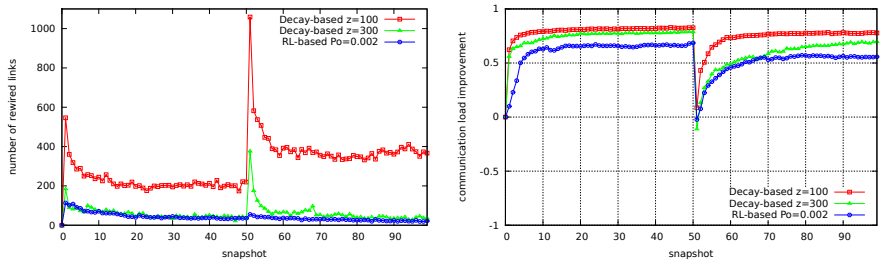


Figure 6.9: (Left) Number of structural relations rewired, and (Right) Communication load improvement when the service demand changes and agents use link-based self-organization mechanisms: `Decay-based` and `RL-based`.

	Number of rewired links					
	$sn = 1$	$sn = 5$	$sn = 50$	$sn = 51$	$sn = 55$	$sn = 99$
$z = 100$	546.2 ± 37.44	289.4 ± 62.38	219.8 ± 78.25	1058.8 ± 113.27	447.0 ± 115.59	366.6 ± 125.29
z = 300	**183.4 ± 41.73**	72.2 ± 20.68	**34.0 ± 14.45**	**376.0 ± 91.58**	85.0 ± 53.86	**35.8 ± 31.39**
$P_o = 0.002$	113.4 ± 28.17	86.0 ± 4.11	36.2 ± 8.16	55.0 ± 10.59	41.0 ± 11.82	22.0 ± 6.78

	Communication load improvement					
	sn=1	sn=5	sn=50	sn=51	sn=55	sn=99
z = 100	0.62 ± 0.02	0.76 ± 0.03	**0.82 ± 0.01**	0.086 ± 0.12	0.64 ± 0.05	**0.77 ± 0.02**
$z = 300$	0.55 ± 0.03	0.68 ± 0.05	0.78 ± 0.02	-0.11 ± 0.47	0.39 ± 0.16	0.69 ± 0.07
$P_o = 0.002$	0.09 ± 0.45	0.54 ± 0.20	0.68 ± 0.039	-0.02 ± 0.17	0.32 ± 0.25	0.55 ± 0.09

Table 6.4: (Up) Number of structural relations changed, and (Down) Communication improvement in different snapshots sn when service demand changes and agents use link-based self-organization mechanisms: `Decay-based`, and `RL-based`.

was able to recover its success rate quickly. The latency of the `Decay-based` mechanism with $z = 100$ was lower than the `RL-based` mechanism. Therefore, we can conclude that the `Decay-based` is more appropriate for dynamic environments.

Figure 6.9 (Left) shows the number of rewired structural relations to deal with varies in the service demand distribution at each moment. For `Decay-based` mechanism with $z = 100$, there was a substantial peak in the number of rewired relationships at the beginning of each interval when the service demand changed. This peak means that agents, considering their local view, were able to be aware that there was a change

in the services that were being demanded in the system, and, therefore, there were links that started to not being used. Agents with parameter $z = 100$ were more prone to changes, and they did not wait very long to decide to rewire these relations that are not being used to more profitable ones with other agents. However, note that the results obtained with $z = 100$ had greater error intervals, which indicates the variability in the number of changes in the structural relations (see Table 6.4). The results obtained with $z = 300$ considerably decreased the number of structural changes and the error intervals since agents wait to receive a higher number of queries before considering the rewiring action. These results show that this configuration also offers a high degree of adaptation. Note that if the change in the service demand is significant (such as in the second interval), the agents note this fact and the number of the structural changes increases accordingly. For the `RL-based` mechanism in our test, the number of structural changes was not as significant as in the `Decay-based` mechanism. This indicates that there is an increase in the latency of the system to achieve a suitable adaptation.

Figure 6.9 (Right) shows the improvement introduced in the communication load of the system with adaptation mechanisms as the service demand changes. This improvement is influenced by the number of structural changes as well as by the path length and the success in the discovery process. For the `Decay-based` mechanism, the best results were obtained with $z = 100$. Note that the results obtained with $z = 300$ were not as good as the results with $z = 100$; however, considering that the number of structural changes was lower, the improvement in the communication load was quite good. For the `RL-based` mechanism, the improvement was not as significant as with the `Decay-based`.

In general, we can conclude that both mechanisms offer good results for self-organization of structural relations between agents. However, the `Decay-based` self-organization mechanism offers better results than the `RL-based` mechanism. The main differences are in the number of structural changes, in the average path length, and in the improvement of the communication load. Moreover, the latency of the system

to achieve a suitable self-organization is greater with the `RL-based` mechanism than with the `Decay-based` mechanism. For highly dynamic environments, the `Decay-based` mechanism is more appropriate.

6.5.2 Population Self-Organization: Leave, Clone, or Remain

In the second set of tests, we evaluated the effects of local `Population-based` mechanism that considers decisions about continuing in the system, leaving the system, or continuing and cloning themselves in order to adapt the population of the system according to the service demand. Initially, agents were distributed uniformly over the set of roles defined in the system. The agents had an average degree of connection of 2.5. There are two reasons to choose this degree of connection. One reason is to evaluate the performance of networks and search algorithms when connection parameters are at limit. The other reason is that could be scenarios where the maintenance of links is costly, therefore, it is interesting to see the behavior of networks when the average number of connections is low. The service demand in these experiments was dynamic. As in the previous test, two intervals where the most demanded services change were defined. In the first interval, there was a set of services that were more demanded than others following an exponential distribution ($\lambda = 0.7$). In the second interval, the least demanded services were the services that were most demanded in the previous interval. Note that this was the worst scenario. Each interval consisted of a set of 50 snapshots. A snapshot contained 10,000 queries. The tables with results are not shown here since the graphs clearly show the results with the error intervals.

Figure 6.10 (Left) shows the effects of local decisions of agents about their situation with respect to the path length of the service discovery process. As the figure shows, the two intervals where there were different service demands are clearly defined by a sharp increase in the path length in the service discovery process. This increase was more significant at the beginning of the second interval since the population of the system was adapted to the opposite service demand. At the beginning of the first

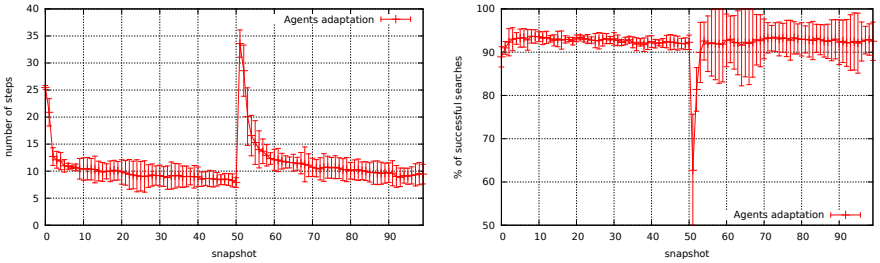


Figure 6.10: (Left)Average number of steps required to locate the provider agent of the required service, and (Right) Percentage of successful queries when agents use `Population-based` self-organization mechanism.

interval, this increase was not as significant since, initially, agents were distributed uniformly over the roles of the system, (i.e., there were the same number of agents that played each role). At the end of the first interval, the distribution of the number of agents per role followed an exponential distribution, where the number of agents that offered services of certain roles was higher than the number of agents that played other roles. For this reason, at the beginning of the second interval, where the service demand followed the opposite service distribution as the first interval, the average path length increased sharply. At that moment, there were only a few agents that offered services that were being demanded, and there was a low probability of locating the required services until the agent population was adapted to the new service demand. Note that the increase of the path length in this scenario was higher than in scenarios where only link-based organization mechanisms were introduced. This is because the degree of adaptation achieved by changing the population of the system was greater, and, therefore, the effects of changes in the service demand had more significant effects. The local decisions of agents about their situation in the system reduced the number of steps required to reach the provider agent needed.

Figure 6.10 (Right) shows the effects of local decisions of agents about leaving or cloning themselves in the service discovery success. The `Population-based` mechanism considerably improved the percentage of successful searches, and it was able to deal with service demand changes. Note that, in the second interval, the

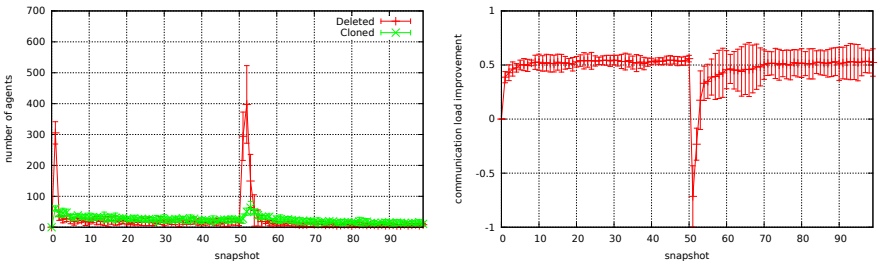


Figure 6.11: (Left) Number of agents that decide to clone or leave the system, and (Right) Communication load improvement as the service demand distribution changes.

error intervals indicate that there was a higher variability in the success rate. This is because the service demand changed to the opposite service demand; therefore, this introduced a higher variability in the system adaptation and consequently in the success of the service discovery process.

Figure 6.11 (Left) shows the number of agents that decided to create a clone or leave the system. It can be observed that, when there was a change in the service demand, the agents that offered some services became aware that these services were not in demand and decided to leave the system. The number of agents that decided to leave the system was greater at the beginning of each interval, and especially in the second interval, since, at the end of the first interval, the majority of the system population was offering services that were demanded in that interval but were now the least demanded. At the beginning of an interval, the decision to clone themselves was taken by a lower number of agents than the decision to leave the system since the decision of this action required the consideration of more information. However, as time passed, the number of agents that decided to clone themselves was higher than the number of agents that decided to leave the system. The trend for both actions (leaving and cloning) fell gradually diminished as the system became adapted.

Figure 6.11 (Right) shows the improvement in the communication load in the system. It can be observed that the self-organization of the population significantly reduced the communication load in the system. The latency to recover the normal behavior of

the system was shorter in the first interval because the agents did not have previous information about a different service demand that introduced noise in their local view. Agents were able to adapt to the service demand in the first 10 snapshots. In the second interval, the latency was greater than in the first interval. The adaptation of the population to the service demand took 20 snapshots. The progress in each interval followed a logarithmic curve, where the greater changes were at the beginning and then the system stabilized.

6.5.3 Combining Self-Organization Strategies

The third set of tests evaluated the combination of the two self-organization mechanisms proposed in this paper: `Decay-based` mechanism with $z = 100$ and `Population-based` mechanism. We selected the value $z = 100$ since we do not consider that structural changes imply a cost. In scenarios where there is a cost in the structural changes, the `Decay-based` mechanism with $z = 300$ is more appropriate. As in the previous test, we evaluated the effects of this combination in a dynamic environment where the service demand changed in each interval. An interval contained 50 snapshots and each snapshot consisted of 10,000 queries. Initially, the agents were uniformly distributed over the organizational roles. The average degree of connection of an agent was 2.5.

Figure 6.12 shows the results related to the path length of the search process. As in previous experiments, there was a sharp increase at the beginning of the second interval; then the number of steps required to locate the provider agent went down and finally remained constant. One of the differences between considering the two mechanisms together or separately was that, at the beginning of each interval, the peak of the number of steps increased more when the two mechanisms were combined. This was because the system achieved a better adaptation to the current service demand, and, therefore, agents required more steps at the beginning of the next interval when the service demand changed. Another difference was the variability in the path length. This variability was reduced when the mechanisms were combined (see Ta-

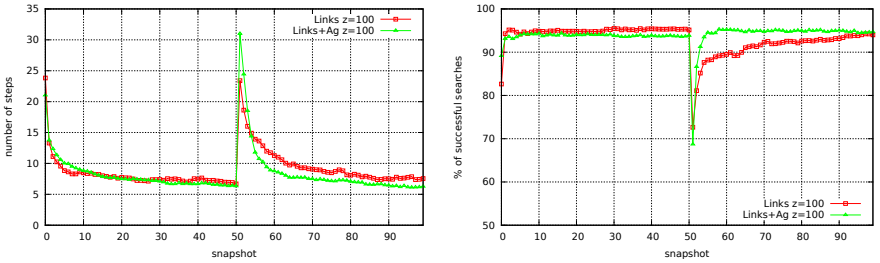


Figure 6.12: (Left) Average path, and (Right) Percentage of successful queries when agents include `Decay-based` and `Population-based` mechanisms in the discovery process.

	Av. path					
	$sn = 1$	$sn = 5$	$sn = 50$	$sn = 51$	$sn = 55$	$sn = 99$
$z = 100$	13.29 ± 1.60	8.84 ± 0.93	6.69 ± 0.6	23.39 ± 3	13.92 ± 2.30	7.56 ± 0.76
$Links + Agz = 100$	13.73 ± 1.5	10.01 ± 1.51	$\mathbf{6.32 \pm 0.89}$	30.98 ± 1.14	11.77 ± 1.73	$\mathbf{6.26 \pm 0.95}$

	% Success					
	$sn = 1$	$sn = 5$	$sn = 50$	$sn = 51$	$sn = 55$	$sn = 99$
$z = 100$	94.28 ± 3.33	94.16 ± 3.73	95.13 ± 0.85	72.64 ± 7.51	88.18 ± 5.56	94.0 ± 1.81
$Links + Agz = 100$	93.10 ± 2.52	93.93 ± 0.91	93.83 ± 1.85	68.7 ± 14.55	94.50 ± 2.05	94.67 ± 2.27

Table 6.5: (Up) Average path length, and (Down) Percentage of successful searches in different snapshots sn when service demand changes and agents include `Decay-based` and `Population-based` mechanisms in the discovery process.

ble 6.5). This is clearly observed in the results obtained in the second interval, where there was a service demand that was completely different to the previous one. This fact is because the adaptation to the service demand of the `Population-based` mechanism reduced the number of rewired structural relations needed. Also, the inclusion of the `Population-based` mechanism reduced the latency of the system to adapt to the new service distribution.

Figure 6.12 (Right) shows the effects of the combination of the two adaptation strategies on the success rate of the service discovery process in the system. At the beginning of an interval where the service demand changed considerably, there was a drop in the percentage of successful searches. The latency of the adaptation at the begin-

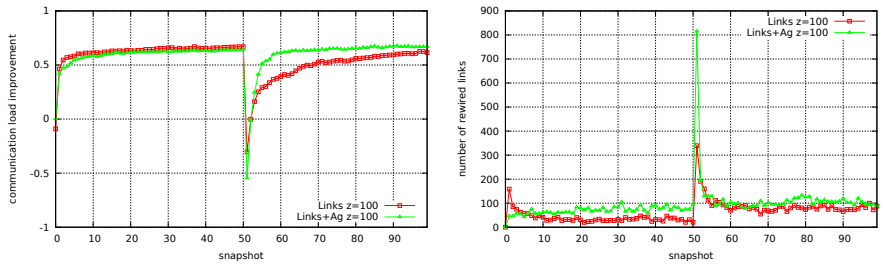


Figure 6.13: (Left) Improvement in the communication load, and (Right) Number of structural changes when `Decay-based` and `Population-based` mechanisms are combined.

ning of an interval was lower when the two mechanisms were combined. The system was able to recover from this situation quickly and achieved a success rate of nearly 95%. This success rate was maintained throughout the entire interval. The drop in the intervals was more significant when the two self-organization mechanisms were combined because the system was more adapted to the service demand. Note that a better system adaptation reduced the variability in the obtained results, mainly when the changes in the service demand were significant (see Table 6.5).

Figure 6.13 (Left) shows the results related to the communication load in the system. In general, the improvement of the communication load when there was a change in the service demand was reduced. However, the system was able to recover and improve the communication cost significantly (around 50%). Note, that the combination of the two strategies reduced variability (see Table 6.6).

With regard to the structural self-organization cost, the combination of the `Decay-based` and `Population-based` methods did not have a significant influence on the number of agents that left the system or cloned themselves. The results obtained are similar to the results obtained with the `Population-based` mechanism that did not consider the link changes (see Figure 6.11 (Left)).

Figure 6.13 (Right) shows the number of changes in the structural relations between agents. The number of structural changes increased when both mechanisms were

	Communication load improvement					
	$sn = 1$	$sn = 5$	$sn = 50$	$sn = 51$	$sn = 55$	$sn = 99$
$z = 100$	0.46 ± 0.12	0.58 ± 0.08	0.66 ± 0.02	-0.30 ± 0.2	0.29 ± 0.15	0.61 ± 0.03
$Links + Agz = 100$	0.42 ± 0.09	0.54 ± 0.04	0.64 ± 0.06	-0.54 ± 0.27	0.51 ± 0.07	0.66 ± 0.05

	Structural changes					
	$sn = 1$	$sn = 5$	$sn = 50$	$sn = 51$	$sn = 55$	$sn = 99$
$z = 100$	158.8 ± 32.66	57.0 ± 30.03	20.6 ± 26.19	339.4 ± 74.57	90.2 ± 46.27	87.8 ± 33.75
$Links + Agz = 100$	46.2 ± 19.53	42.2 ± 23.23	95.4 ± 32.55	815.2 ± 144.02	129.4 ± 45.59	87.6 ± 114.41

Table 6.6: (Up) Communication load improvement, and (Down) Number of structural changes in different snapshots sn when service demand changes and agents include `Decay-based` and `Population-based` mechanisms in the discovery process.

combined. This is due to the fact that the system achieved a greater degree of adaptation that included not only the structural links, but also the population of the system; therefore, a change in the service demand required a higher number of structural changes.

6.6 Conclusions

Environmental conditions in open, service-oriented systems change and the systems should be able to adapt to new circumstances in a decentralized way and taking only local information into account. This chapter addresses the problem of adaptation to changes in the environment through self-organization mechanisms. In the presented system, neither the agent population nor the structural relations between agents remain constant. Agents in the system include two self-organization mechanisms in order to adapt the system to changes in the service demand.

On one hand, agents estimate the utility of their links by analyzing the number of times they use each link to forward queries. The weight of each link decays with time if it is not being used. The relationships with neighbors that are not being used are placed with new structural relations with acquaintances. The acquaintances are established as a results of the service discovery activity.

On the other hand, agents are also able to estimate whether or not they are playing an important role in the system. This estimation is carried out by calculating their structural homophily in the system. Based on their structural homophily, agents decide to remain, leave, or clone themselves in order to adapt the population of the system to the service demand.

We evaluated the proposed approach through a set of experiments taking into account the effects of the inclusion of self-organization mechanisms of the average path length, the percentage of successful searches, the improvement in communications and the time to recover from changes. We analyzed the behavior of each self-organization mechanism both separately `Decay-based` and `Population-based` and combined.

The `Decay-based` mechanism offers a more reactive self-organization. Therefore, the latency or time for adaptation is reduced considerably with respect `RL-based` mechanism. However, this reactive behavior implies an adaptive cost in the number of structural relations changed. The number of changes in the structural relationships between agents can be regulated through the displacement parameter z. The experiments show that the best results are obtained with $z = 100$. However, in systems where the structural changes are expensive, the configuration with $z = 300$ reduces the structural changes and also offers good results.

We compared the proposed `Decay-based` mechanism with a `RL-based` mechanism. The best configuration of `Decay-based` mechanism offers better results than the best configuration of `RL-based` mechanism. `RL-based` mechanism reduces the number of structural changes, and, therefore, takes more time for adaptation. This mechanism is not suitable for highly-dynamic environments.

The `Population-based` mechanism is also appropriate for self-organization in distributed environments. The degree of adaptation obtained with this mechanism is higher than with `Decay-based` mechanism. This causes changes in the service demand to have more significant effects. However, this mechanism also offers a small latency to recover from changes in the service demand.

The combination of `Decay-based` and `Population-based` mechanisms offers a more dynamic system adaptation. This combination improves previous mechanisms in the percentage of successful searches, the average path, and communication load. The latency to adapt to the new service demand is also reduced as well as the variability in the obtained results. The improvement introduced with the combination of both mechanisms is more significant when drastic changes in the service demand are produced and agents have outdated information in their internal states.

In conclusion, the proposed model, which integrates self-organization mechanisms, can be used in the context of service discovery in open Service-Oriented MAS to improve the performance of the system under dynamic changes. Agents are able to reason about when it is the most appropriate to apply self-organization actions. The proposed mechanisms are based on local information and allow agents to continuously adapt their structural relations and their situation in the system.

CHAPTER

7 Cooperation Emergence in Service-Oriented MAS

7.1 Introduction

There are distributed systems where the cooperation of all the entities that participate in them is required to obtain a good performance that provides benefits for all the participants. Some of the scenarios where cooperation is required are: wireless ad-hoc networks where nodes rely on other nodes to forward their packets in order to reach the destination node; file sharing in P2P systems [133]; streaming applications [80], discussion boards [53], on-line auctions [121], or overlay routing [15]. If participants that decide not to contribute in order to maximize their own benefits and exploit the contributions of the others appear in these scenarios, they will obtain a high rate of benefits in the short term. However, these benefits decrease as the number of selfish participants increases, thereby damaging the performance of the whole system.

There are models of genetic and cultural evolution that confirm that the opportunity to take advantage of others undermines and often eliminates cooperation [57]. These cooperation problems are also known as social dilemmas (i.e., the tragedy of the commons, the free-rider problem, the social trap). The promotion and stabilization of cooperation in scenarios of this type has been considered to be an area of interest [42].

One of the scenarios where cooperation plays an important role is service discovery in open Service-Oriented Multi-Agent systems (Service-Oriented MAS) [38]. These systems are populated by agents that offer their functionality through services [84]. Agents are social entities that are aware of other agents. However, sometimes this awareness is not enough to find potential collaborators in order to achieve the goals of agents. Therefore, open Service-Oriented MAS should provide mechanisms to facilitate the discovery of services provided by other agents. Nevertheless, this is not an easy task due to the intrinsic characteristics of these systems. Agents only have a partial vision of the system, and there is no a central control that is responsible for supervising the system. As a consequence, agents need the cooperation of their neighbors in order to forward queries to locate the required resources or services. Moreover, this becomes even more difficult when there are self-interested agents that do not cooperate with other agents in order to avoid the cost of forwarding queries. In that case, if there are no mechanisms to deal with these agents and promote cooperation, the performance of the whole system could be seriously compromised.

This chapter proposes a combination of decentralized mechanisms to facilitate the emergence of cooperation in a service discovery scenario. In this scenario, agents are located in a network and their interactions are influenced by the network structure. We propose the integration of *social plasticity* and the use of *incentives* to promote cooperation when self-interested agents appear. *Social plasticity* is used as a structural punishment to those agents that decide not to cooperate in repeated encounters. *Incentives* are used to promote the cooperation among agents during the service discovery.

The main contributions of the proposal presented in this chapter are the following: (i) our mechanism is asynchronous, i.e., agents update their behavior when they participate in the service discovery process; (ii) the payoff calculation is based on local information obtained from the activity of agents and the results in the discovery process; (iii) in the rewiring process, agents only break links with those neighbors that have non-cooperative behavior, and instead of replacing them randomly, the agents look for another agent based on their preferences; (iv) agents are able to detect when it is more appropriate to use incentives or social plasticity taking into account local information about the degree of cooperation of their neighborhood.

The chapter is structured as follows. Section 7.2 describes the service discovery scenario where cooperative and non-cooperative agents are present. Section 7.3 explains three mechanisms to promote cooperation in service-oriented environments. The first is based on local structural changes in the structure of the network. The second is based on incentives. The third mechanism is the combination of the two previous mechanisms. Section 7.4 presents a set of experiments where we evaluate the performance of the proposed mechanisms and compare them with other proposals. Finally, section 7.5 presents conclusions and final remarks.

7.2 Cooperation in Service Discovery Scenario

Distributed systems rely on the cooperation of the entities that participate in them. However, in open and heterogeneous environments, a common and more realistic situation is that selfish agents appear in them [29]. Agents of this type exploit the cooperation of other agents and deny their cooperation in order to maximize their benefits. If the number of agents that decide not to cooperate with others increases, the efficiency of the system could be seriously damaged.

In the scenario of Figure 7.1a we show an scenario where all the agents are cooperative. Agent a_i can forward the query to agent a_k and the discovery process only requires 3 steps to locate the provider agent a_v that offers the service it needs. In the

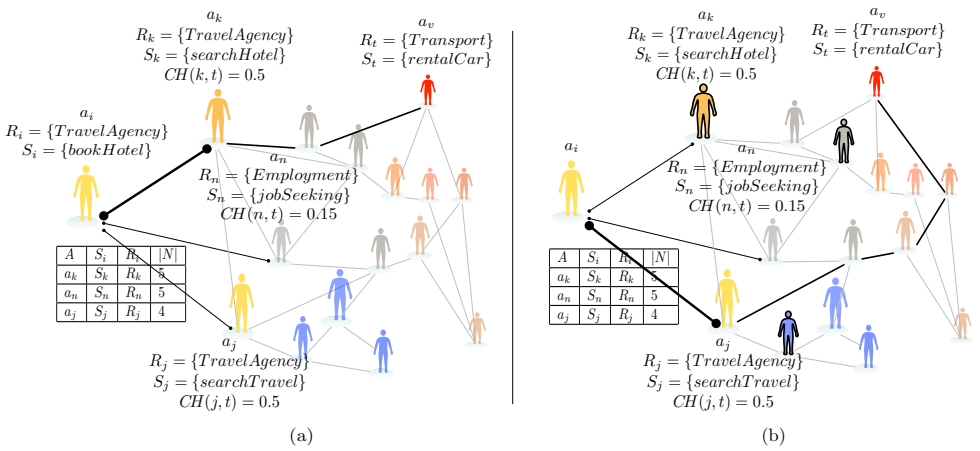


Figure 7.1: Service discovery scenario.(a) All the agents are cooperative. Agent a_i only knows its direct neighbors a_k, a_j, and a_n. If a_i needs to locate a service (i.e., $rentalCar$), it will forward the query to its most promising neighbor (i.e., a_k) based on the homophily between the neighbor and the target agent (i.e., a_t) that should provide the required service and the degree of the neighbor. (b) Agents with thick silhouette are not cooperative. Agent a_i has to find an alternative agent to a_k since a_k refuses to forward queries.

scenario of Figure 7.1b there are cooperative agents that collaborate with others in the forwarding process, and agents, such as a_k (thick silhouette), that are not cooperative and decide not to forward queries from other neighbors compromising the service discovery process. As an example, if agent a_i receives a query about a service that it does not offer, it should choose one of its neighbors. Agent a_i considers that the most suitable agent to forward the query is a_k. However, agent a_i should find another neighbor, a_j, to forward the query since the behavior of a_k is non-cooperative and it is not going to forward the query. As a result of this non-cooperative behavior, the number of steps required to reach the target agent a_v increases (5 steps).

In this particular scenario of service discovery, if the number of agents that decide not to forward queries from other agents increases, the length of the paths and the messages generated in the discovery process increases considerably since non-cooperative agents should be avoided and alternative paths should be found. Therefore, cooperative agents must forward more queries, the number of searches that end

successfully is reduced, and the system efficiency decreases. The worst case is when the length of a path is near to the TTL and the service is finally not found. In this situation, the effort of all cooperative agents is useless. For that reason, it is important to provide mechanisms to be able to confront situations where agents are pursuing their own goals without cooperating and are compromising the performance of the overall system [70]. These mechanisms should promote and maintain cooperation in the system in order to guarantee the proper performance of the system [42].

In the following sections, we describe the information that agents consider when it is appropriate to deal with non-cooperative behavior, and two mechanisms that facilitate the emergence of cooperation. One of them is based on local structural changes. The other one is based on the use of incentives. We also describe under which circumstances are more appropriate to use them. Finally, we explain how both mechanisms can be combined to improve the performance of the service discovery and maintain the degree of cooperation even in scenarios where the predominant behavior is not collaborate.

7.3 Promoting Cooperation

The structural relations between agents influence their interactions, and, therefore, influence the behavior of the agents. In the discovery process, if an agent needs to locate another agent that provides a service in order to achieve one of its goals, it should rely on the cooperation of its direct neighbors. Based on the local information about the success or failure of their previous interactions with their direct neighbors, agents update their behavior (i.e., cooperate or not cooperate in the discovery process) and decide when it is appropriate to change their current structural relations.

In order to facilitate agents to reason about their behavior in the service discovery process, each agent maintains a local view of the activity in the system. This local view is stored in the internal state of the agent st_i (see Definition 2). Regarding the reasoning process about cooperation, agents consider the information contained in

their internal states, specifically, in their model of the local environment $\mathcal{K}_i^E$ that has been extended from the definition presented in Section 6.3. The information from the $\mathcal{K}_i^E$ that the agents use during the cooperation analysis is the following:

- dc_i represents the degree of cooperation of agent a_i. dc_i ranges in the interval [0,1]. Agents are not completely cooperative or non-cooperative. They have a degree of cooperation that establishes the probability of cooperate.
- $\mathcal{C}_i$ represents the behavior of agent a_i in the last interaction with other agents. $\mathcal{C}_i$ can take two values: cooperative or not cooperative,
- $\mathcal{Q}_i$ is the number of queries that agent a_i forwarded,
- $\mathcal{SQ}_i$ is the number of queries that the agent a_i forwarded in discovery processes that ended successfully,
- $\mathcal{RQ}_{ij}$ is the number of queries from agent a_i that agent a_j refused to forward,
- $\mathcal{P}_i$ is the number of service requests attended to by agent a_i,
- $\mathcal{R}_i$ is the number of service requests created by agent a_i.

Based on the local information that agents have in their internal state (st_i), they can decide to deal with non-cooperative neighbors using social plasticity, incentives, or the combination of both.

7.3.1 Structural Mechanism: Social Plasticity

Structure is an important feature to consider in the cooperation models [79]. The structure of the network influences interactions of agents, therefore it is important to provide agents mechanisms to be able of changing their local structure in the network. We propose the inclusion of a structural mechanism. Through interactions during the service discovery process, agents are able to change their relations taking into account

which neighbors provide profitable relationships and which do not. This feature is called *social plasticity* [44]. Social plasticity is the capacity of individuals to change their relationships as time passes. Specifically, in our system, each agent maintains information related to its neighbors. This information consists of the number of times a neighbor has refused to forward one of its queries $\mathcal{RQ}_{ij}$. The agent keeps a counter for each of its links in its internal state (st_i). Each counter is increased by one unit each time that a query is refused by a neighbor. If a neighbor decides to change its behavior and forwards queries, the agent updates the counter to 0. Therefore, a cooperative agent that occasionally cannot forward queries, because its workload at a certain moment is too high to attend to more queries is not considered to be a non-cooperative agent.

Each time an agent tries to forward a query to one of its neighbors a_j, it updates its information associated to the link with a_j and evaluates the utility of the link. In order to evaluate the utility of a link, an agent a_i uses a decay function that calculates the probability of maintaining a link with a_j taking into account the number of queries that it would have sent through neighbor a_j but that a_j refused to forward. This function is a sigmoid that ranges between [0,1],

$$D(\mathcal{RQ}_{ij}) = \frac{1}{1 + e^{\frac{-(\mathcal{RQ}_{ij} - d)}{y}}}, \tag{7.1}$$

where $\mathcal{RQ}_{ij}$ is the number of queries that neighbor a_j received from agent a_i and that a_j decided not to forward. The constant y is the slope and d is the displacement. These constants are established by the agent. The parameter y indicates if the transition between considering a structural change or not is going to be sharp or gradual. As the value of y increases, the transition is more gradual. The displacement d indicates how benevolent an agent is with respect the non-cooperative behavior of its neighbors. The increase of the value of d means that the agent is going to consider a higher number of refuses in order to make a decision about looking for another neighbor. The decrease of the value of d means that it is not permissive with the number of refuses (see Figure 7.2). The function $D(\mathcal{RQ}_{ij})$ returns a value in the range [0,1],

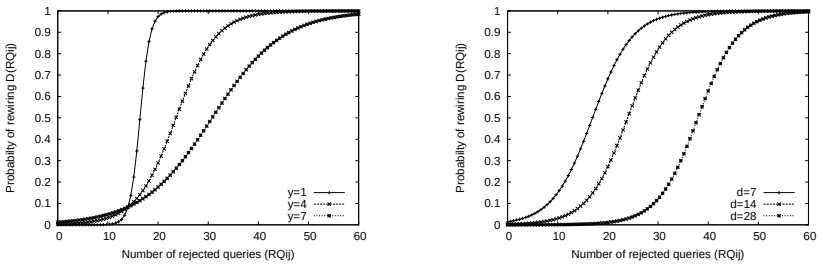


Figure 7.2: Sigmoid function that calculates the probability of breaking a current link with a neighbor and looking for a new neighbor. The figure shows the shape of the function with different values of slope parameter y and displacement parameter d.

Algorithm 10 Function that an agent uses to analyze if it is appropriate to change a link.

1: **function** $socialPlasticity(a_i, a_j)$
2: **if** $D(RQ_{ij}) > (random(0,1))$ **then**
3: $N_i \leftarrow N_i - \{a_j\}$
4: $newAg \leftarrow lookForNeighbor(a_i)$
5: $N_i \leftarrow N_i \bigcup \{newAg\}$
6: $N_{newAg} \leftarrow N_{newAg} \bigcup a_i$
7: **end if**
8: **end function**

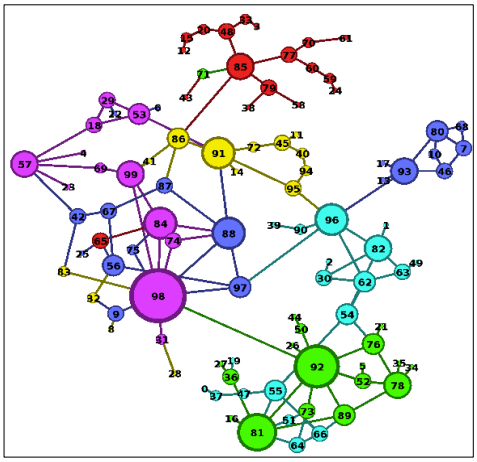
where 0 indicates that the agent does not consider that the number of rejects from its neighbor is enough to make a decision about rewiring, and 1 indicates that it is necessary to change the link.

Algorithm 10 shows the *socialPlasticity* function where an agent a_i evaluates its link with one of its neighbors a_j. The agent a_i uses the D function and considering this probability it decides to break its current link and look for a neighbor to maintain its degree of connectivity (Alg. 10 Line 4). We assume that any alternative agent (co-operator or non cooperator) always accepts a new partner. There are different criteria for establishing a new link with another agent in the network. We have considered two criteria:

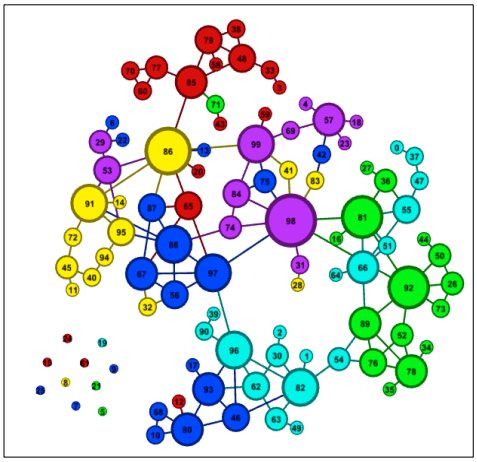
- establishing a link with a *neighbor's neighbor*;
- establishing a link based on *choice homophily*.

To illustrate how these criteria affect the system structure, let us begin with simple examples of networks where agents have social plasticity to ostracize agents that are not cooperating. Figure 7.3 shows the effect of ostracizing non-cooperative agents with different rewiring mechanisms. Each network has 100 agents. Each agent is represented by a node and plays a role and offers a semantic service associated to the role. There are 6 roles defined in the system. Each color represents the role that an agent plays. The number inside a node is the identifier of the agent. Agents are distributed uniformly over the roles and services in the system. The average degree of connection of the network is 2.5. Figure 7.3a shows the initial structure of the system. Note that the effect of homophily makes agents establish a higher number of connections with similar agents than with dissimilar ones. The number of non-cooperative agents is 25. An agent that breaks a link with another agent looks for a new one. As stated above, we consider two criteria to replace a link with a non-cooperative neighbor. We analyzed the effects of each criterion after 1,000 queries that were uniformly generated by the agents:

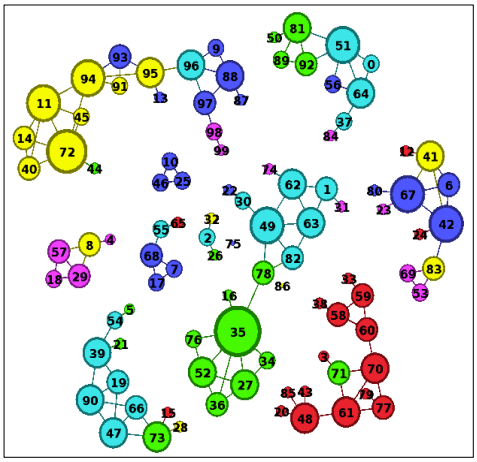
Neighbor's neighbors. The effects of using the criteria *Neighbor's neighbors* are shown in Figure 7.3b. The non-cooperative agents are the agents with id-labels that range in the interval [0,25]. The nodes that are non-cooperative are isolated as a result of the local decisions of the agents. The structure of the network remains connected and the success rate of the discovery process does not change. However, the structure of the network does not always remain connected. In this case, the non-cooperative agents have a low degree of connection and they do not are not located in a critical position in the structure of the network (i.e., hub that connects different communities of agents). A different situation is when non-cooperative agents have a high degree of connection. In Figure 7.3c, non-cooperative agents are those nodes with id-labels that range in the interval [99,74]. These agents with a high degree of connection play an important role in connecting different communities. The ostracism of these agents using the *Neighbor's neighbors* criterion not only disconnects or reduces the connectivity of non-cooperator agents but also divides the network into several isolated



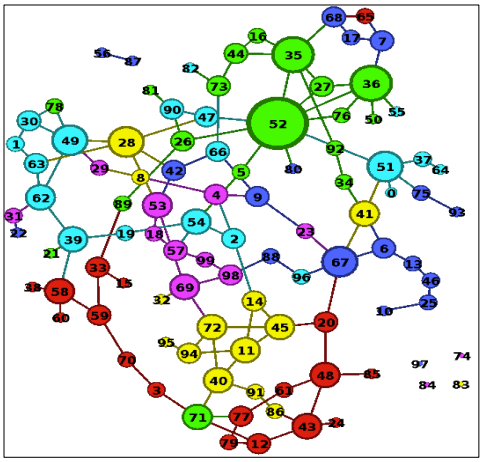
(a) Original system structure.



(b) Structure of the system when non-cooperative agents have a low degree of connection. The rewiring criterion of the agents is 'Neighbor's neighbors'.



(c) Structure of the system when non-cooperative agents have a high degree of connection. The rewiring criterion of the agents is 'Neighbor's neighbors'.



(d) Structure of the system when non-cooperative agents have a high degree of connection. The rewiring criterion of the agents is 'choice homophily'.

Figure 7.3: Effects of social plasticity on the structure of the system where 25 agents are non-cooperative and 75 cooperative.

parts. On one hand, the rewiring strategy increases the degree of clustering inside the community. On the other hand, the rewiring strategy decreases the probability of establishing connections with different communities, thereby decreasing the success rate of the search process since services from different communities cannot be reached.

Choice Homophily. The effects of the mechanism based on the probability of establishing a link with an agent that offers similar services and plays similar roles to the previous neighbor are shown in Figure 7.3d. The probability of agent a_i establishing a connection with an agent similar to a_j is proportional to the homophily between the candidate neighbor and the current neighbor ($CH(a_j, a_n)$). This strategy gives cooperative agents the opportunity to maintain their inter-community connections avoiding the creation of isolated communities. As Figure 7.3d shows, this mechanism allows agents to maintain the structure of the network even though non-cooperative agents are located on a critical position in the structure of the network. The non-cooperative agents (nodes with id-labels that range in the interval [99,74]) have lost all their connections or their degree of connection has been considerably reduced. Therefore, they lose their privileged position in the network and now are located on the fringes of the network or completely ostracized.

Therefore, we consider that the *Choice homophily* criterion is suitable to avoid the system disconnection when social plasticity is considered by the agents and non-cooperative agents have a high degree of connection (i.e., act as hubs in the service discovery process). The maintenance of the connected system is important to maintain the success rate of the discovery process since services located in far positions could be reached.

The use of social plasticity to isolate or to reduce the degree of connection (thereby the influence of non-cooperative agents) improves the performance of the system. However, the use of structural mechanisms when the number of non-collaborator agents increases could break the network structure into several isolated parts, thus

reducing the system performance. Moreover, it is difficult to break links with non-cooperative agents that are located in the fringes of the network and have a low degree of connection. These non-cooperative agents do not receive enough queries to consider a rewiring action. Note that, in scenarios where structural changes have a significant cost, this mechanism can not be always profitable. As an alternative to this method, in the following section, we propose an incentive mechanism that does not change the network structure to facilitate the emergence of cooperation.

7.3.2 Incentive Mechanism

In the proposed model, agents can have cooperative or non-cooperative behavior. Cooperating in the service discovery scenario implies that an agent is going to: forward queries, request services, and attend to requests about its services. If an agent has non-cooperative behavior, it means that the agent is going to act selfishly by requesting services and offering its services, but it is not going to forward the queries that it receives from its neighbors.

We assume that each action in our model implies a cost or a benefit. For instance, forwarding a request has a cost since an agent has to dedicate time and resources to decide which neighbor is the best one to forward the query to. If a query finally arrives to an agent that provides the required service (i.e., the search process ends before the TTL), then the agents that participated by cooperating in the forwarding process will obtain a reward for their contribution. Otherwise, the agents lose their investment in the forwarding process. Moreover, an agent that locates the required provider agent must pay for the service and the provider gets a benefit for attending to the request.

Agents are rational entities that update their own behavior to maximize their own payoff. An agent calculates its payoff with the following function based on its behavior

and taking into account its actions:

$$\mathcal{PO}(st_i) = \mathcal{SQ}_i \cdot sq - \mathcal{Q}_i \cdot q + \mathcal{P}_i \cdot p - \mathcal{R}_i \cdot r \tag{7.2}$$

where:

- $\mathcal{SQ}_i$ (number of queries forwarded in discovery process that ended successfully), $\mathcal{Q}_i$ (number of queries forwarded), $\mathcal{P}_i$ (number of services provided), $\mathcal{R}_i$ (number of requests generated) is the information of the internal state (st_i) of an agent,
- q is the cost of forwarding queries,
- sq is the benefit obtained by the agents that participate by forwarding queries in a service discovery process that ends successfully,
- p is the benefit obtained by the agents that provide a service,
- r is the cost of requesting a service.

We assume that all the agents have the same costs and benefits for the actions. The strategy update rule implemented in this model is based on imitation [132]. Agents take into account the payoff of their direct neighbors to update their behavior. If an agent has a neighbor that obtains a higher payoff, the agent changes its behavior to the behavior of its neighbor (see Algorithm 11).

We would like to remark that cooperative behavior analysis is asynchronous. Agents that analyze and update their cooperative behavior are those that during the service discovery process are considered to be the most promising candidates to forward a query, even though they finally do not forward the query.

When the number of cooperative agents is greater than the number of non-cooperative agents, non-cooperative agents are prone to change their behavior to cooperate since the probability that a query ends successfully is high, and, therefore, cooperation

Algorithm 11 Function that updates the behavior of an agent taking into account the payoff of its neighbors.

```
1: function CooperationAnalysis(a_j)
2: payOff_j ← PO(st_j)
3: neighborToImitate ← ∅
4: for a_n ∈ N_j do
5:     if PO(st_n) > payOff_j then
6:         neighborToImitate ← a_n
7:     end if
8: end for
9: C_j ← C_neighborToImitate
10: dc_j ← dc_neighborToImitate
11: return C_j, dc_j
12: end function
```

receives a reward if the discovery process ends successfully. However, when the number of non-cooperators is greater than the number of cooperators, cooperative behavior does not always emerge. In this case, the incentive of cooperating in the discovery process is not always enough to maintain cooperation.

To asses the impact of the incentive mechanism, we conducted several simulations in small networks of 100 agents and two different configurations. In the networks of Figure 7.4, 75% of the agents were cooperative and 25% were non-cooperative. The rest of the structural parameters of the networks were similar to the networks presented in Figure 7.3. The costs and benefits of the actions were: $q = 0.15$, $sq = 0.30$, $p = 0.5$, and $r = 0.5$. Agents update their behavior when they participate in the discovery process. In the networks of Figure 7.4, non-cooperative agents are represented by red nodes and cooperative agents by blue nodes.

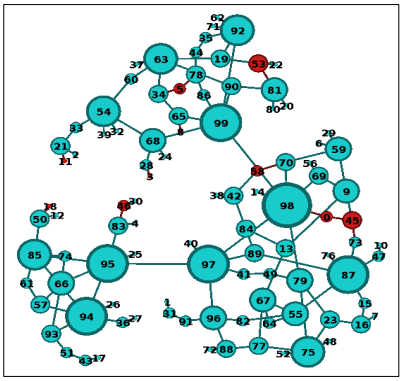
Figure 7.4a shows an example of the effects of the incentive mechanism after 1,000 queries generated in the system where the degree of connection is uniformly distributed over the agents without taking into account their behavior. In this scenario, the incentive mechanism is enough to promote cooperation among agents. The majority of agents that do not cooperate are situated on the fringes of the network since these positions are not easily influenced.

Figure 7.4b shows an example of the effects of the incentive mechanism after 1,000

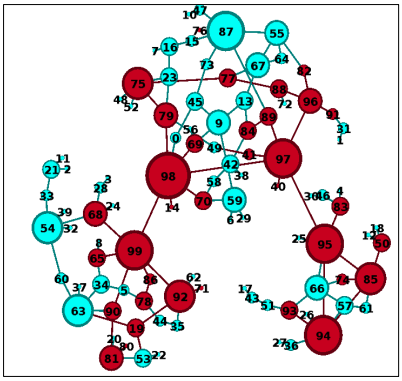
queries generated in the system where the non-cooperative agents had a high degree of connection. The non-cooperative agents got benefits quickly since they received a high number of service requests and they do not have the cost of forwarding others' queries. The cooperative agents had a great number of losses when agents with a high degree of connection did not cooperate because the discovery process took more steps, and, therefore, cooperative agents had the cost of forwarding queries but they had a low probability of receiving a benefit since the number of non-cooperators was too high and the probability of being successful decreased significantly. In this scenario, non-cooperative agents obtained a higher payoff than cooperative agents, and, therefore, had a greater influence on their neighborhood. Although the influence of the non-cooperative agents was clear, their influence was not enough to convert all the cooperative agents into non-cooperative agents. There are some special situations where cooperative agents have influence over the non-cooperative even though they have a low degree of connection. These cooperative agents are located on the fringes of the network with a degree of 1. Therefore, they have less probability of participating in the search process, and they do not have many losses because of the forwarding process. This fact gives them more benefits than their neighbors, and they can influence their behavior. Moreover, nodes that have a neighborhood with the same behavior and a low degree of connectivity do not change their behavior. However, this is not enough to influence nodes beyond the neighborhood.

7.3.3 Adaptive Combination of Social Plasticity and Incentives

The use of structural mechanisms such as social plasticity or incentives promotes the emergence of cooperation. On one hand, social plasticity allows agents to isolate or reduce the connectivity and therefore the influence of those non-cooperators that are exploiting the system and specially those non-cooperators that have a high degree of connection. However, if social plasticity is the only mechanism used and the number of non-cooperators is higher than the number of cooperators, the network could be broken into several isolated parts which seriously affects to the system performance.



(a) Initially, the system was populated by a set of 25 non-cooperative agents and 75 cooperative agents. The degree of connection was uniformly distributed over agents. The figure shows the effects of using incentives to promote cooperation among agents.



(b) Initially, the system was populated by a set of 25 non-cooperative agents that had a high degree of connection (agents with ids in the range [99,75]). The figure shows the distribution of cooperation after the use of incentives to promote cooperation among agents.

Figure 7.4: Evolution of the number of cooperative and non-cooperative agents after 1,000 queries generated in the system. Initially, there were 75 agents that cooperate and 25 that did not cooperate. Agents consider incentives during the service discovery process. The nodes represent agents and the numbers are the identifiers of the agents. Blue nodes represent cooperative agents and red nodes non-cooperative agents.

On the other hand, a mechanism based on incentives tries to motivate a change in the behavior of agents to cooperate without changing the structure of the network. However, if highly connected nodes are non-cooperative or the number of non cooperative agents increases, then the expected payoff does not compensate the effort to cooperate.

We propose the integration of both mechanisms in order to combine their advantages to deal with non-cooperative scenarios. Incentives is used to change the behavior of non-cooperative agents and to be more effective is combined with social plasticity. Social plasticity is used by agents when the majority of their neighbors are non cooperative in order to reduce the influence of the non-cooperative ones. The influence decrease of non-cooperative agents makes more effective the use of incentives. With the combination of both mechanisms the social plasticity reduces the influence of non-cooperator agents and this fact increases the effectiveness of incentives.

Algorithms and describes a modified version of the service discovery process Algorithms 2 and 3 presented in Chapter 5 to include the social plasticity and incentives. Algorithm 14 shows how these two mechanisms have been combined and integrated in the service discovery process. Specifically, both mechanisms are included in the process carried out by each agent when it has to decide the most promising neighbor to forward the query to the target agent. Once an agent a_i has selected a neighbor a_j, agent a_j evaluates its behavior comparing its payoff with the rest of its direct neighbors (Line 7). Based on this comparison, the agent decides whether or not to change its behavior in order to improve its payoff in future interactions.

If agent a_j does not cooperate, then a_i increases the number of times its neighbor a_j has refused to forward a query. Moreover, a_i evaluates whether or not it is appropriate to rewire the current link with a_j in order to find a better connection. In order to find a trade-off between the number of structural changes and the emergence of cooperation, the use of the social plasticity mechanism is affected by the number of non-cooperator agents that an agent has in its neighborhood (Lines 11-12). If the number of non-cooperator neighbors is over a certain threshold, the mechanism used to facilitate the emergence of cooperation is the social plasticity mechanism combined with the mechanism based on incentives. Otherwise, the mechanism used is based on incentives only. If a_j does not cooperate, a_i eliminates a_j from the set of neighbors to consider in the current forwarding process and repeats the process until a cooperative neighbor is found or the set of neighbors to consider is empty (Lines 13 and 19).

Finally, if a_i finds a neighbor that cooperates, the neighbor increases by one its local information about the number of queries forwarded (Line 17). Otherwise, the search process fails.

With the combination of the two mechanisms, social plasticity and incentives, non-cooperative agents lose connectivity, benefits, and influence in the neighborhood. As a consequence, they decide to change their behavior to the most promising behavior in the neighborhood, which is to cooperate. In situations where the number of

Algorithm 12 Function that describes how an agent a_i starts a service discovery process.

function $startDiscovery()$
$\mathcal{R}_i \leftarrow \mathcal{R}_i + 1$ /* increase of the number of requests generated by a_iM ODIFICATION */
$TTL \rightarrow 100$ /* number of steps associated to the query */
$part \rightarrow \{\}$ /* list of participants in the service discovery process MODIFICATION */
$\varepsilon \rightarrow 0.75$ /* threshold similarity */
$a_j, part \leftarrow serviceDiscovery(part, a_i, q = (a_i, s_q, r_q, TTL, \varepsilon))$ /* MODIFICATION */
/* if a suitable provider agent is found before the TTL */
if $a_j \neq \emptyset$ **then**
    $request(a_j, s_q)$
    $\mathcal{P}_j \leftarrow \mathcal{P}_j + 1$ /* increase of the number of services provided by agent a_j START MODIFICATION */
    for $a_n \in part$ **do**
        $\mathcal{SQ}_n \rightarrow \mathcal{SQ}_n + 1$ /* increase of the number of queries that agent a_n forwarded in a discovery process that ends successfully */
    end for /* END MODIFICATION */
end if
end function

Algorithm 13 Function that describes the service discovery process that an agent a_i carries out when it receives a query.

function $serviceDiscovery(part, a_i, q = (a_{source}, s_q, r_q, TTL, \varepsilon))$
$found \leftarrow false$
$a_t \leftarrow (r_q, s_q, \emptyset, \emptyset)$ /* target agent that represents the profile of a suitable provider agent to solve the query */
if $TTL > 0$ **then**
    /* if an agent enough similar to the target is found */
    if $CH(a_i, a_t) \geq \varepsilon$ **then**
        $found \leftarrow true$
        $inform(a_{source}, part, a_i)$
    else
        /* if the agent is not enough similar, it selects the most promising neighbor */
        $a_i \leftarrow promisingNeighbor(a_i, a_t)$ /* START MODIFICATION */
        if $a_i \neq \emptyset$ **then**
            $part \rightarrow part \cup \{a_i\}$
            $TTL \leftarrow TTL - 1$
            $serviceDiscovery(part, a_i, q = (a_{source}, s_q, r_q, TTL, \varepsilon))$
        else
            $inform(a_{source}, \emptyset, \emptyset)$
        end if /* END MODIFICATION */
    end if
else
    $inform(a_{source}, \emptyset, \emptyset)$
end if
end function

Algorithm 14 Function that describes how an agent selects the most promising neighbor when heterogeneous agents that cooperate and do not cooperate are considered.

function $promisingNeighbor(a_i, a_t)$
$N'_i \leftarrow N_i$ /* list of candidate neighbors to consider to forward the query */
$numCoop \leftarrow 0$ /* number of cooperative neighbors */
$coop \leftarrow false$
repeat
  $a_j \leftarrow \mathcal{F}_{N'_i}(a_t)$ /* Most promising neighbor selection function (see Formula 5.8) */
  $\mathcal{C}_j, dc_j \leftarrow cooperationAnalysis(a_j)$
  /* if the neighbor a_j does not cooperate, a_i considers the use of social plasticity */
  if $dc_j < random(0,1)$ **then**
    $\mathcal{RQ}_{ij} \leftarrow \mathcal{RQ}_{ij} + 1$ /* increase of the number of queries that a_j rejects from a_i */
    $numCoop \leftarrow coopNeighbors(a_i)$ /* $numCoop$ number of cooperative neighbors of agent a_i */
    /* if the number of cooperative neighbors of agent a_i is under a threshold ($N_i \cdot \rho$) */
    if $coop < N_i \cdot \rho$ **then**
      $socialPlasticity(a_i, a_j)$
    end if
    $N'_i \leftarrow N'_i - \{a_j\}$ /* the neighbor a_j is deleted from the list of candidates to consider to forward the query */
    $coop \leftarrow false$
  else
    $coop \leftarrow true$
    $\mathcal{Q}_j \leftarrow \mathcal{Q}_j + 1$ /* increase of the number of queries that a_j forwarded */
  end if
until $\neg coop \vee (|N'_i| == 0)$ /* until a cooperator neighbor is found or the list of candidates is empty */
if $(|N'_i| == 0)$ **then**
  return $\emptyset$
else
  return a_j
end if

non-cooperative agents is significant, this mechanism allows the emergence of cooperation. Some agents remain non-cooperative because they are located where the degree of clustering and the degree of connection are too low; therefore, the number of services provided and the queries forwarded are too low to influence the others.

7.4 Evaluation

In this section we evaluate the proposed mechanisms to promote cooperation in service-oriented environments. First, we evaluate the social plasticity mechanism. Specifically, we analyze the behavior of the mechanism with different configuration parameters and in different networks. Second, we test the behavior of the incentives mechanism when different reward values are used in different network configurations. Finally, we evaluate the combination of both mechanism and we compare it with other approaches used to promote cooperation in distributed environments.

In order to see the effects of the introduction of heterogeneous agents with different degrees of cooperation on the performance of the service discovery system, we conducted several simulations. In these simulations, the networks had 1,000 agents and there were 16 different roles defined in the system. Each agent played one role and offered one service associated to the role it played. The set of semantic service descriptions used for the experiments has been taken from the OWL-S TC4 test collection [1]. Roles and services were distributed following a uniform distribution. The degree of cooperation of an agent ranged in the interval [0,1]. The value assigned to ε parameter was 0.75. The average degree of connection of the agents was 2.5.

All the agents in the system had the same probability of generating service queries. A query consisted of two features that characterize the required provider agent: the role and the service. A query was successfully solved when an agent that offered a similar service (i.e., the degree of semantic match between the semantic service descriptions

[1] http://www.semwebcentral.org/projects/owls-tc/

was over a threshold $\varepsilon = 0.75$) was found before the TTL ($TTL = 100$). The query distribution in the system was modeled as a uniform distribution.

7.4.1 Social Plasticity

In order to see the influence of the slope parameter y and the displacement parameter d (see Figure 7.2), we tested the behavior of the service discovery system when agents incorporate social plasticity with different values of y and d. The tests were done in 10 networks with 1,000 agents where there were 600 non-cooperative agents and 400 cooperative agents. Each snapshot consisted of 5,000 queries.

Figure 7.5 shows the results of the test. The rows show the structural changes, the average path to reach the target agent, and the success in the discovery process. The columns represent each configuration value for the displacement parameter d ($d = 7, d = 14, d = 28$). In each configuration of the parameter d we have evaluated the influence of parameter y ($y = 1, y = 4, y = 7$).

The first row shows the total number of structural changes in the system in each snapshot. When agents were configured with $d = 7$, they were less benevolent with their non-cooperative neighbors and the number of structural changes was larger than in the other configurations. Note that in all configurations the highest number of changes were in the first iterations. The effect of a high number of structural changes is that the system isolates the number of non-cooperators earlier. As the value of parameter d increases, the number of structural changes is reduced. Note that values 1 and 4 of parameter y also reduced the number of structural changes.

The number of structural changes make that the success rate in configurations with values $d = 7$ and $d = 14$ increases earlier than with other configurations with values $d = 28$ and the final success rate when the system is adapted is higher with configurations $d = 7$ and $d = 14$ than configuration $d = 28$. In the success results the parameter y has not a significant influence.

In general, the average path increases as the non-cooperative agents are isolated. At the first iterations, the only searches that ended successfully were those that located a suitable provider agent in a few steps because, in short paths, there was a lower probability of finding a non-cooperator agent. In the following snapshots, the agents isolated the non-cooperative agents and the path length of successful searches increased. This happened because not only queries about services provided by nearby agents could be solved, but also queries about services that were provided by agents located outside of the circle of interaction of the agents that initiated the discovery process could be solved. Note that configurations $d = 7$ and $d = 14$ obtained similar results of average path length. In each configuration, the main differences were established by the value of the parameter y. However, the configuration $d = 14$ made a lower number of structural changes than $d = 7$. Therefore, for scenarios where structural changes have a cost associated to them, the configuration of decay function $D(\mathcal{RQ}_{ij})$ for evaluating the links of an agent with $d = 14$ is better than $d = 7$. Note that the results indicate that the transition between considering a structural change or not should be sharp ($y = 1$) rather than gradual ($y = 7$, $y = 14$) since it reduces the structural changes and the average path length.

We analyzed the social plasticity criterion to find a replacement of a neighbor that it is not cooperating. Specifically, we considered *Neighbor's Neighbors* and *Choice homophily* (CH). We also evaluated how the increase of non-cooperative affects the success rate and the number of steps required in successful searches. For the experiments we consider two types of networks: networks where cooperative and non-cooperative agents had the same probability of having a high degree of connection, and networks where non-cooperative agents had a high degree of connection. In the experiments, 25,000 queries were uniformly generated by the agents. The parameters of the social plasticity mechanism were $d = 14$ and $y = 1$.

Figure 7.6 shows in the first row the success rate of the queries and in the second row the average path length of the successful queries. Left column shows the results obtained in networks where all the agents had the same probability of having a high

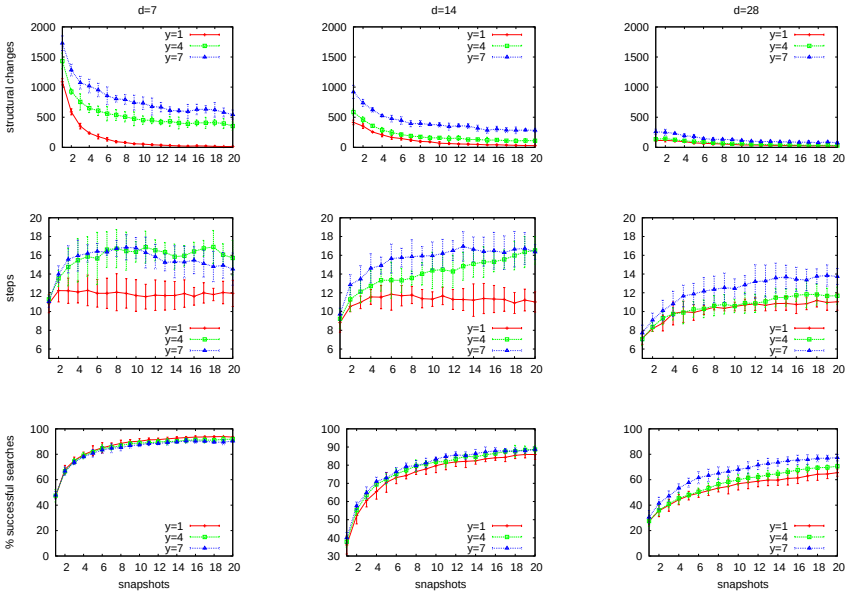


Figure 7.5: Influence of slope parameter y and displacement parameter d in a service discovery system where there are 600 non-cooperative agents and 400 cooperative agents.

degree of connection. Right column shows the results obtained in networks where non-cooperators had a high degree of connection. The x-axis shows the percentage of non-cooperative agents.

Regarding the success rate results (see first row of Figure 7.6), as expected, in systems with a static structure, where agents could not break links with non-cooperative agents, the percentage of success in the service discovery process decreased considerably. In dynamic systems, providing the agents with social plasticity to modify their relations benefits the search process. If the number of non-cooperative agents increases, agents with social plasticity isolate non-cooperative agents and although the success rate decreases, the decrease is not as significantly as in static networks. The search process in dynamic networks obtained a success rate of over 80% in the case where 40% of agents were not collaborating in forwarding tasks. This was an increase of 40% over the success rate obtained in static networks. We can conclude that

the use of social plasticity to isolate non-cooperative agents benefits system performance. However, the social plasticity that used the criterion *CH* offered better results than the social plasticity that used *Neighbor's Neighbors* criterion. The reason is that *Neighbor's Neighbors* criterion divides the network into isolated parts that reduces the probability of reaching the required agent. The difference between these two criteria is more clear in networks where non-cooperative agents have a high degree of connection (see Figure 7.6 right column).

The second row of Figure 7.6 shows how the path length of queries that were successfully solved changed as the number of non-cooperative agents increased. In static networks where all agents had the same probability of having a high degree of connection the path length of successful queries as the number of non-cooperative agents did not change significantly. The reason is that only queries that were generated close to the agent that provided the service could be solved. In the case of dynamic networks, the path length increased when agents used the criterion *Neighbor's Neighbors*. This is because agents were able to find alternative paths avoiding non-cooperative agents. When agents used the criterion *CH*, the path length of the successful queries remained around 11 steps since with the *CH* criterion the network structure was maintained and therefore the average path length.

In static networks where non-cooperative agents had a high degree of connection the path length increased until the 15% of non-cooperators. This increase was because the non-cooperative agents blocked short paths and agents should find alternative paths that took more steps. When the percentage of non-cooperators was over the 15% the success rate decreased and only queries that could be solved in the neighborhood ended successfully. Something similar happened to dynamic networks where agents used social plasticity with the *Neighbor's Neighbor* criterion or the *CH* criterion. However, these effects were less significant in the case of networks where agents used social plasticity with *CH* criterion.

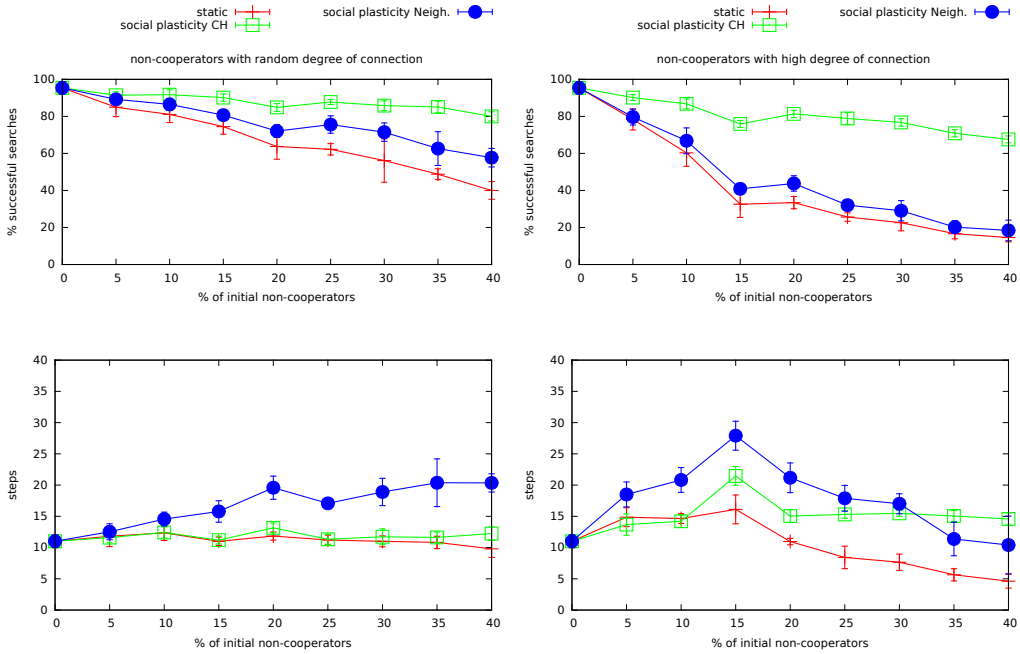


Figure 7.6: Performance evaluation of social plasticity mechanism as the number of non-cooperative agents increases in two types of networks: (Left column) networks where all agents had the same probability of having a high degree of connection, (Right column) networks where non-cooperator had a high degree of connection. Each row shows: % of queries that end successfully, and average number of steps in service discovery.

7.4.2 Incentives

We analyzed the effects of different rewards to agents that cooperate in successful discovery processes in the success rate, in the average path length of successful searches, and in the emergence of cooperative behavior in the system. We considered two types of networks: networks where cooperative and non-cooperative agents had the same probability of having a high degree of connection, and networks where non-cooperative agents had a high degree of connection. In the experiments 75,000 queries were generated in 10 different networks of each type. The costs and benefits of the actions were: $q = 0.15$ cost of forwarding action, $p = 0.5$ benefit of providing a service, and $r = 0.5$ cost of asking for a service. We considered three values for rewards sq:

- 0.15 that is a reward that compensates the cost of the forwarding action ($q = 0.15$) but the agents do not obtain any extra reward,
- 0.20 that is a reward that compensates the cost of the forwarding action and agents obtain an extra reward of 0.05,
- 0.30 this reward that compensates the cost of the forwarding action and agents obtain an extra reward of 0.15.

The results are shown in Figure 7.7. In networks where all the agents had equal probability of having a high degree of connection the rewards $sq = 0.15$ and $sq = 0.20$ were not enough to promote cooperation. Although the incentives mechanism with reward $sq = 0.20$ did not promote cooperation, the success rate was not seriously affected until the network had the 25% of the initial agents with a non-cooperative behavior. In the case of networks where non-cooperative agents had a high degree of connection, the rewards $sq = 0.15$ and $sq = 0.20$ could not promote cooperation and the reward $sq = 0.30$ promoted cooperation only in networks where the percentage of initial non-cooperator was not over the 10%. The presence of a high number of non-cooperators made a decrease in the success rate and in the average

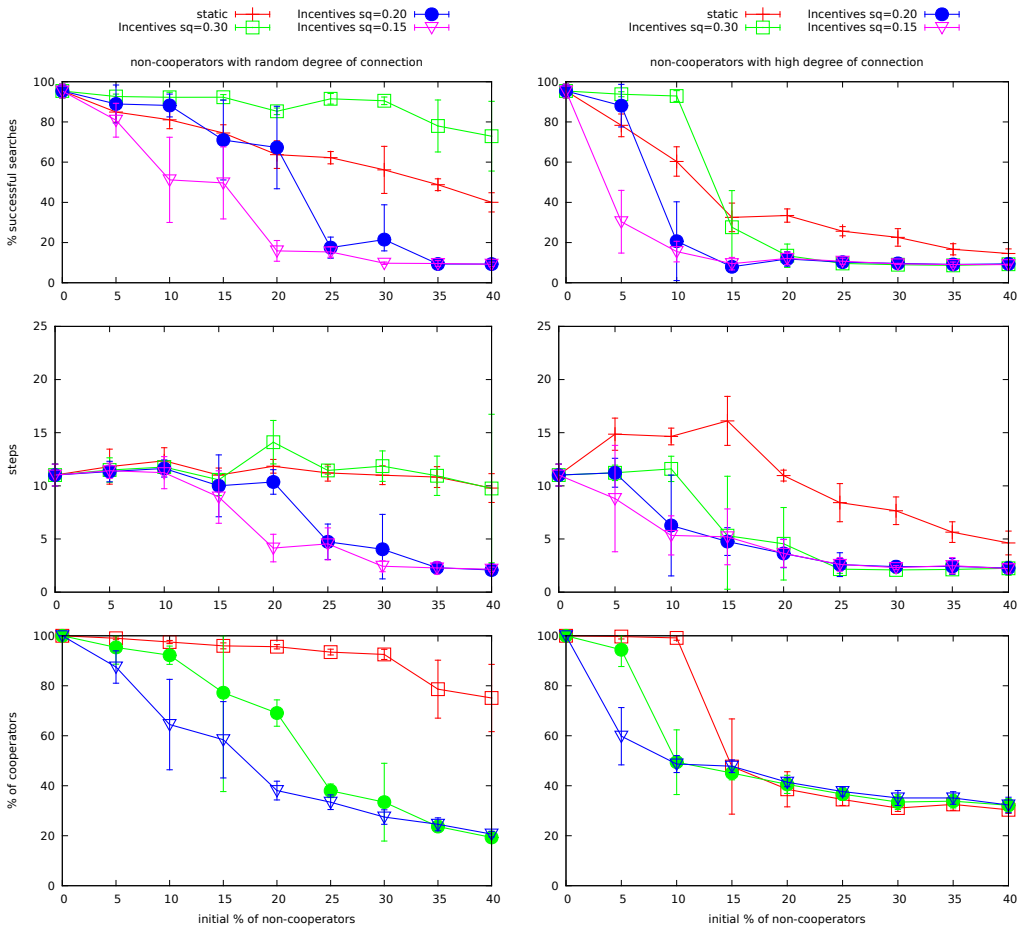


Figure 7.7: Performance evaluation of incentives mechanism with different rewards as the number of non-cooperative agents increases in two types of networks: (Left column) networks where all agents had the same probability of having a high degree of connection, (Right column) networks where non-cooperator had a high degree of connection. The first row shows % of queries that end successfully. The second row shows the average number of steps in service discovery. The third row shows the number of cooperator agents.

mean path of the successful searches since only those queries that could be solved in the neighborhood of the agent that generated the query.

	C	NC
C	(1, 1)	(0, 1.5)
NC	(1.5, 0)	(0, 0)

	C	NC
C	(0.15, 0.15)	(-0.15, 0)
NC	(-0.15, 0)	(0, 0)

Figure 7.8: Payoff matrix: (Left) Prisioner's Dilemma [44], (Right) Stag Hunt Dilemma.

7.4.3 Social Plasticity and Incentives

Several tests were performed to evaluate the effects of the combination of social plasticity and incentives to facilitate the emergence of cooperation in a decentralized service discovery system. In the experiments, we made a snapshot of all of the metrics every time 5,000 queries were solved in the system in order to see the evolution of the metrics. We compared our proposal of combining social plasticity and incentives with the separate use of both mechanisms. Moreover, we also compared the proposal with other mechanisms present in the literature. The set of approaches that we considered in the tests were the following:

- **Social plasticity (SP)**: agents only consider social plasticity to promote cooperation in the system. The value used for the displacement parameter in the decay function was $d = 7$ and the value used for the slope parameter was $y = 4$.
- **Incentives**: agents only consider incentives to facilitate the emergence of cooperation. The costs and benefits of the actions were: $q = 0.15$, $sq = 0.30$, $p = 0.5$, and $r = 0.5$.
- **Incentives and Social Plasticity (Incentives+SP)**: agents consider the combination of incentives and social plasticity to facilitate the emergence of cooperation. The costs and benefits of the actions were: $q = 0.15$, $sq = 0.30$, $p = 0.5$, and $r = 0.5$, and the value for the displacement parameter was $d = 7$ and the value used for the slope parameter was $y = 4$.
- **Reinforcement Learning (RL)**: Reinforcement learning has been used as an

approach for solving decision-making problems in multi-agent systems. It allows agents to dynamically adapt to changes without requiring global knowledge. The reinforcement learning method used to promote cooperation was WPL. This algorithm is based on the following idea to achieve convergence: slow down learning when moving away from a stable policy and speed up when moving towards a stable policy [2]. This idea is similar to the Win or Lose Fast (WOLF) method [18], but the WPL method offers better performance than WOLF. The WPL has been adapted to the context of service discovery. Each agent a_i maintains two matrices, π_i and Q_i, with two dimensions $|st| \times |acc|$ (states and actions). There are two possible states in the context of cooperation: cooperate or not cooperate. There are two possible actions considered in the context of service discovery are: forward or refuse to forward a query. The matrix π_i stores the probability of cooperating. The value $\pi_i(st, acc)$ represents the degree of cooperation of an agent in a certain state. The value $Q_i(st, acc)$ represents the expected reward an agent will obtain by executing an action when it is in a certain state; this value is based on the number of agents in the neighborhood of agent a_i that collaborated and the number of queries that agent a_i sent and were solved successfully. Moreover, each time an agent participates in the service discovery process it considers the re-organization of its links (add or remove links) based on its degree of connection and the initial average degree of connection of the network.

- **Game Theory.** We considered two type of games: the Prisioner's Dilemma (**PD**) [44], where individuals might not cooperate even though it seems to be their best interest to do so; and Stag and Hunt (**SH**) [69], which describes a conflict between safety and social cooperation. The main difference between them is the payoff matrix (see Figure 7.8). In these games, cooperate implies forwarding queries and not cooperate rejecting forward queries. These games do not consider the actions of requesting services or providing services.

  An agent a_i plays the PD (or the SH) with its neighbors (a_j) when it is considered as the most appropriate agent to forward the query. The process of

updating the behavior of the agents is similar to the process described in [44]. Agent a_i and each agent in its neighborhood ($a_j \in N_i$) calculate their aggregate payoff within their neighborhood. Agent a_i updates its current strategy by imitating the strategy of the neighbor with the largest payoff. If a_i imitates a non-cooperative agent a_j, it breaks its link with a_j and establishes a new link with another agent taking into account the homophily criterion with a probability of $p = 0.1$.

To evaluate the 'Incentives+SP' mechanism and compare it with the other approaches, we considered two scenarios: (i) one in which the number of cooperators was greater than the number of non-cooperators, and (ii) one in which the number of non-cooperators was greater than the number of cooperators. Specifically, the tests in each scenario focus on a set of metrics that are meaningful for the analysis of the performance of the system: (i) the evolution of the number of cooperator agents in the system; (ii) the average number of steps required to locate an appropriate agent that solves a query; (iii) the percentage of queries that are solved before the TTL; (iv) the number of failures caused by the presence of non-cooperator agents.

Each graph has an associated table that contains the results with error intervals obtained in certain snapshots sn. The results obtained in the first snapshots ($sn = 1$ and $sn = 5$) are shown since there were more significant differences between mechanisms and configurations than in other snapshots. The last snapshot $sn = 20$ is also shown since it reflects the final results when the system evolved. For reasons of clarity, the error intervals of the results are not shown here in the graphs.

7.4.3.1 Scenario with 600 cooperators and 400 non-cooperators.

In scenarios where the number of cooperator agents is greater than the number of non-cooperator agents, the majority of the mechanisms improve the degree of cooperation in the system (see Figure 7.9 (Left)). The best degree of cooperation was achieved by the mechanism based on 'SH' and the mechanism based on 'Incentives'. The

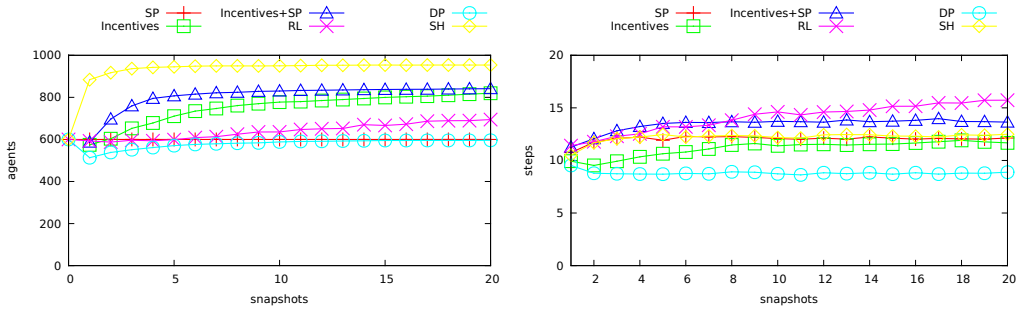


Figure 7.9: Evaluation of (Left) the evolution of cooperation in the system, and (Right) the average path length in the discovery process when there are 600 cooperative agents and 400 non-cooperative agents.

	Collaboration			Av. Path		
	$sn = 1$	$sn = 5$	$sn = 20$	$sn = 1$	$sn = 5$	$sn = 20$
SP	600	600	600	10.74 ± 0.61	11.92 ± 0.67	12.13 ± 0.73
Incentives	574.2 ± 107.89	710.6 ± 321.2	819.8 ± 322.3	9.92 ± 0.63	10.64 ± 2.76	11.67 ± 2.80
Incentives+SP	587.0 ± 161.3	807.2 ± 90.3	841.4 ± 54.9	11.27 ± 0.46	13.51 ± 1.14	13.64 ± 1.30
RL	592.8 ± 12.8	598.8 ± 35.5	694.4 ± 27.4	11.40 ± 0.7	13.2 ± 0.91	15.72 ± 1.4
DP	513.0 ± 83	570.6 ± 69.7	595.8 ± 56.1	9.52 ± 0.53	8.69 ± 0.64	8.88 ± 8.88
SH	884.2 ± 31.04	945.4 ± 23.2	953.6 ± 20.3	8.88 ± 0.73	12.46 ± 1.15	12.49 ± 1.38

Table 7.1: (Left) Collaboration and (Right) average path length in different snapshots sn when agents use different strategies to promote cooperation during the discovery process.

'SH' mechanism achieved a high degree of cooperation since the initial degree of cooperation made agents to prefer risk and cooperate than safety and not cooperate. 'Incentives' and 'Incentives+SP' also achieved a high degree of cooperation since in this scenario the incentive for the forwarding action was worth it in the majority of service discovery process. However, the 'Incentives+SP' achieved a lower degree of cooperation than 'SH' due to the social plasticity isolated a high number of non-cooperator agents that could not change their behavior.

The average number of steps of queries that were successfully solved is shown in Figure 7.9 (Right). The largest average path length appeared with the 'SP' mechanism. This is because it took more time to isolate the non-cooperative agents, and, therefore, the forwarding process had to avoid non-cooperative agents and required more steps. Moreover, with 'SP' the number of potential provider agents is reduced by the ostracism of the non-cooperators. The mechanism 'Incentives+SP' offered a lower number of steps than 'SP' since the influence of incentives facilitated changes in the agent's behavior and promoted cooperation. The 'SH' and 'Incentives' mechanisms offered shorter paths than 'Incentives+SP' because the degree of cooperation achieved was nearly 100%; therefore, the number of queries that could only be solved by agents located far away with this mechanisms could be solved and the success in the discovery process increased. The 'PD' mechanism obtained shorter paths than the other mechanisms since the number of non-cooperative agents did not decrease and this made that the only successful searches were those that could be solved by a nearby agent. Therefore, the average path length of successful searches was the shortest. The error intervals of the results in certain snapshots are shown in Table 7.1. In general, the error intervals increase as time passes due to networks have a higher number of structural changes and there is more variability in the structure of the networks.

Figure 7.10 (Left) shows the number of searches that failed because to an agent could not find a neighbor that forwarded a request. The mechanisms that reduced the number of failures for non-cooperation were 'SP' and 'Incentives+SP'. This is because

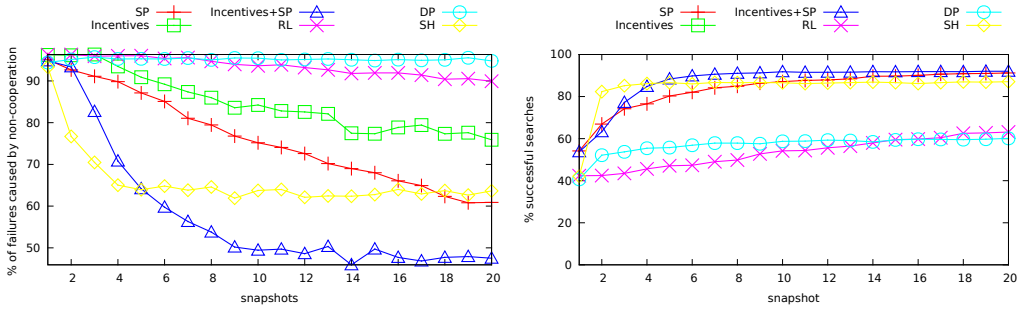


Figure 7.10: Evaluation of (Left) the percentage of failures because of the absence of cooperation, and (Right) the percentage of searches that end successfully in the system when there are 600 cooperative agents and 400 non-cooperative agents.

	Failures			Success		
	$sn = 1$	$sn = 5$	$sn = 20$	$sn = 1$	$sn = 5$	$sn = 20$
SP	95.1 ± 0.84	87.1 ± 3.4	60.8 ± 7.5	54.14 ± 5.34	80.26 ± 4.05	91.23 ± 2.69
Incentives	96.2 ± 1.4	90 ± 13	75 ± 25	43.67 ± 4.10	65.35 ± 41.46	79.55 ± 35.72
Incentives+SP	94.7 ± 1.9	64 ± 24	47 ± 13	53.74 ± 4.90	88.36 ± 9.61	92.02 ± 3.96
RL	96.2 ± 0.7	96.0 ± 0.9	89 ± 3	42.36 ± 3.81	47.09 ± 8.86	63.14 ± 7.2
DP	94.4 ± 0.8	95 ± 1	95 ± 1	40.63 ± 6.40	55.75 ± 12.65	60.14 ± 11.90
SH	93.2 ± 1.3	60 ± 10	63 ± 9	42.12 ± 3.97	86.59 ± 3.67	87.08 ± 3.78

Table 7.2: (Left) Searches failed because of the absence of cooperation and (Right) successful searches in different snapshots sn when agents use different strategies to promote cooperation during the discovery process.

	Str. Changes		
	$sn = 1$	$sn = 5$	$sn = 20$
SP	274.6 ± 52.4	78.4 ± 14.2	13.0 ± 6.3
Incentives+SP	254.8 ± 43.7	23.4 ± 10.6	
RL	220.8 ± 49.8	38.6 ± 15.1	12.4 ± 5.7
DP	488.6 ± 156.4	27.2 ± 10.5	0.6 ± 0.9
SH	6.4 ± 2.6		

Table 7.3: Structural changes in different snapshots sn when agents use different strategies to promote cooperation during the discovery process.

both mechanisms isolated the non-cooperators quickly. 'Incentives' and 'SH' reduced the initial percentage of failures in the first snapshots and then remained constant because the number of non-cooperators are located in the fringes of the network or in an strategic position that makes difficult change their behavior; therefore, the number of failures for the absence of cooperation remained constant. When the 'RL' and the 'DP' mechanisms were used, the number of non-cooperators could not be sufficiently reduced; therefore, the main reason for failure was non-cooperation. In general, the percentage of failures decreases but the error intervals associated to the data increase as time passes due to the changes in the behavior of the agents and the network structure introduce variability in the failure results (see Table 7.2).

Figure 7.10 (Right) shows the percentage of searches that were solved successfully. The strategies of 'SP', 'Incentives', 'Incentives+SP', and 'SH' obtained good results. The main difference among them is the number of iterations required to achieve a successful rate near 90%. The mechanism that offered the best results was the 'Incentives+SP' because when the degree of cooperation in the local neighborhood increases, agents only consider incentives and do not use social plasticity; therefore, the number of agents that remain connected and can be considered to participate in the discovery process is higher than with other mechanisms such as 'SP'. The error intervals in certain snapshots are shown in Table 7.2

In Figure 7.11, we compare the number of structural changes made by the agents using the mechanisms for cooperation. The combination 'Incentives+SP' reduced the number of structural changes required to achieve cooperation if we compare the results with the 'SP' strategy. This is important because there are scenarios where changes in the structure entail a cost and also because the excessive use of 'SP' reduces the number of potential provider agents to consider, and therefore, reduces the possibilities of finding a suitable provider in the service discovery process. The 'DP' strategy generated the highest number of structural changes since agents did not cooperate and there was a high number of links between non-cooperative agents that were broken in the first iterations. The 'SH' strategy generated only a few structural

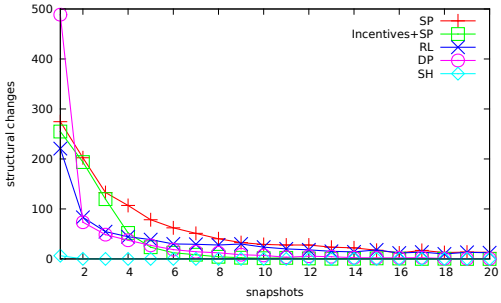


Figure 7.11: Number of structural changes in the system because of the social plasticity of the agents when there are 600 cooperative agents and 400 non-cooperative agents.

changes since agents preferred the risk of cooperate and this promoted cooperation in the network decreasing the number of links between non-cooperative agents and therefore the number of structural changes. The 'RL' strategy generated less structural changes than 'Incentives+SP' and were not enough to promote the cooperation in the network. In general, the error intervals of the results decrease as time passes and the behavior of the agents in the network stabilizes and non-cooperative agents are isolated (see Table 7.3).

In this scenario, there are not significant differences between the mechanisms. In general, 'SP', 'Incentives+SP', and 'SH' offer the best results. However, the 'SP' and 'Incentives+SP' are able to reduce the number of failures for not cooperating in the discovery process. The main differences between 'SP' and 'Incentives+SP' are the average path length since the 'SP' reduces the number of potential provider agents and also the number of structural changes to maintain the proper performance of the system.

7.4.3.2 Scenario 400 cooperators and 600 non-cooperators.

In scenarios where the number of non-cooperators is greater than the number of cooperators, the mechanisms to facilitate the emergence the cooperation become more

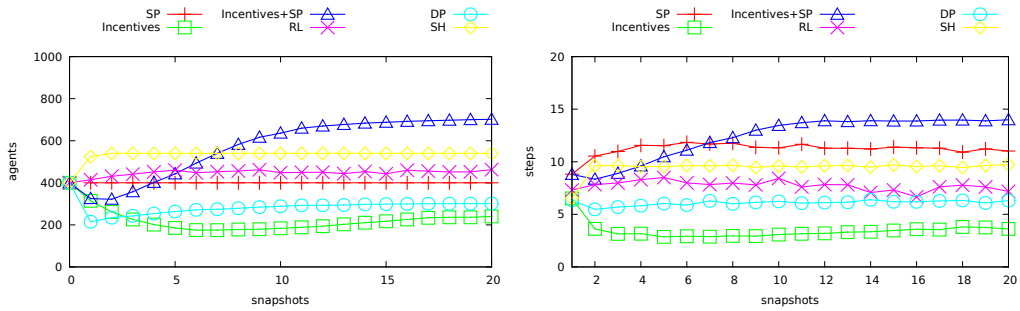


Figure 7.12: Evaluation of (Left) the evolution of cooperation in the system, and (Right) the average path length in the discovery process when there are 400 cooperative agents and 600 non-cooperative agents.

important. The behavior of the system when 600 non-cooperator and 400 cooperator agents are present in the system is evaluated. In this scenario, the differences between the mechanisms are greater than in the previous scenario since there is a greater number of non-cooperator agents.

Figure 7.12 (Left) and Table 7.4 show the evolution of cooperation in the system when different mechanisms were used by the agents to promote cooperation. The best results were obtained by the 'Incentives+SP' mechanism. 'Incentives+SP' achieved the cooperation of the majority of agents in 5 snapshots. The 'SH' mechanism obtained worse results than 'Incentives+SP' mechanism due to the presence of a high number of non-cooperators. Agents prefer the safety of not cooperating over the risk of cooperating. The 'RL' mechanism maintained the cooperation level, but it could not increase it. The 'DP' mechanism did not promote cooperation due to the payoff for not cooperating taking into account that the majority of the network did not cooperate. Therefore, the agents imitated the behavior of the agents with the highest payoff which were the non-cooperators. The 'Incentives' mechanism did not promote cooperation since the number of agents that did not cooperate forced the cooperators to invest resources in searches that were going to fail; therefore, the payoff of the cooperators decreased and the non-cooperators did not imitate them.

Figure 7.12 (Right) shows the average path length of successful searches. The 'SP'

	Collaboration			Av. Path Length		
	$sn = 1$	$sn = 5$	$sn = 20$	$sn = 1$	$sn = 5$	$sn = 20$
SP	400	400	400	8.82 ± 1.03	11.53 ± 1.23	11.01 ± 1.08
Incentives	314.2 ± 15.22	185.4 ± 54.1	240.2 ± 145.4	6.51 ± 0.55	2.85 ± 2.55	3.61 ± 5.88
Incentives+SP	324.8 ± 53	442.4 ± 42.4	702.0 ± 106.7	2.85 ± 0.83	14.06 ± 1.12	13.99 ± 1.23
RL	414.4 ± 10.1	460.2 ± 48.7	462.4 ± 32	7.30 ± 0.86	8.5 ± 1.24	7.18 ± 1.62
DP	214.8 ± 128.5	263.4 ± 148.9	301.0 ± 171.22	6.33 ± 1.01	6.03 ± 3.08	6.30 ± 3.32
SH	524.2 ± 157.37	539.8 ± 128.7	539.8 ± 128.7	6.53 ± 0.52	9.54 ± 1.05	9.72 ± 1.24

Table 7.4: (Left) Collaboration and (Right) Average path length in different snapshots sn when agents use different strategies to promote cooperation during the discovery process.

and 'Incentives+SP' mechanisms obtained longer paths because the number of potential provider agents was reduced since some of them could not be reached because they were isolated. Consequently, there were some service provider agents that took more steps to find. In the rest of the mechanisms, the number of non-cooperator agents was high enough to make the majority of the searches fail. Hence, the number of queries that ended successfully was low and these queries were those that could be solved near the neighborhood of the agent that generated the query.

Figure 7.13 (Left) and Table 7.5 show the percentage of failures caused by non-cooperator agents. The 'Incentives+SP' and 'SP' mechanisms reduced the number of failures considerably. Since the other mechanisms could not deal with non-cooperators, the main reason for the unsuccessful searches was the absence of cooperation. Figure 7.13 (Right) shows the percentage of successful searches. When agents used the 'Incentives+SP' or 'SP' mechanisms, cooperator agents were able to deal gradually with non-cooperators and improved the successful rate of searches. The 'SH' mechanism also improved the success rate, but the improvement was not as significant as the improvement achieved by 'Incentives+SP' or 'SP'.

Finally, Figure 7.14 and Table 7.6 compare the number of structural changes made by the different mechanisms. As in the previous scenario, agents were able to detect at the first snapshots that structural changes are necessary to promote cooperation. In this scenario, the mechanism 'Incentives+SP' made more structural changes than 'SP' since in the first iterations with the mechanism 'Incentives+SP' the number of

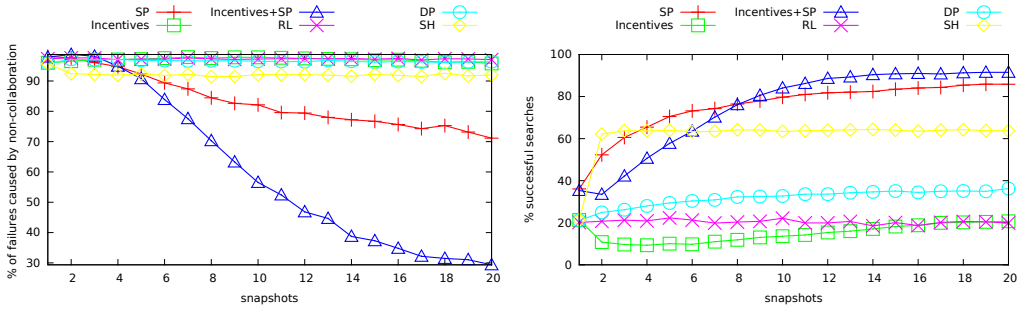


Figure 7.13: Evaluation of (Left) the percentage of failures because of the absence of cooperation, and (Right) the percentage of searches that end successfully in the system when there are 400 cooperative agents and 600 non-cooperative agents.

	Failures			Success		
	$sn = 1$	$sn = 5$	$sn = 20$	$sn = 1$	$sn = 5$	$sn = 20$
SP	98 ± 0.96	92 ± 3.15	71 ± 11	36.08 ± 4.99	70.53 ± 5.45	85.86 ± 2.74
Incentives	95 ± 0.78	97 ± 1.1	95 ± 4.7	21.3 ± 1.84	10.04 ± 5.81	20.69 ± 10.43
Incentives+SP	97 ± 0.96	90 ± 19	29 ± 7	35.36 ± 3.40	57.74 ± 10.86	91.52 ± 1.57
RL	97 ± 0.57	97 ± 0.75	97 ± 0.75	20.16 ± 1.86	22.32 ± 2.24	19.88 ± 3.36
DP	95 ± 1.2	96 ± 1.1	96 ± 1.9	21.10 ± 1.75	29.39 ± 12.14	36.24 ± 18.22
SH	95 ± 1.1	92 ± 2.6	91 ± 3.8	21.37 ± 3.15	63.94 ± 10	63.69 ± 10.32

Table 7.5: (Left) Searches that failed because of the absence of cooperation and (Right) searches that end successfully in different snapshots sn when agents use different strategies to promote cooperation during the discovery process.

	Str. Changes		
	$sn = 1$	$sn = 5$	$sn = 20$
SP	414.2 ± 38.11	164.6 ± 30.68	28.8 ± 10.32
Incentives+SP	397.4 ± 44.77	247.4 ± 26.51	1.2 ± 2.28
RL	183.4 ± 25.90	55.2 ± 7.58	18.2 ± 3.99
DP	468.4 ± 200.57	59.2 ± 62.05	59.2 ± 3.99
SH	14.4 ± 7.29	0 ± 0	0 ± 0

Table 7.6: Structural changes in different snapshots sn when agents use different strategies to promote cooperation during the discovery process.

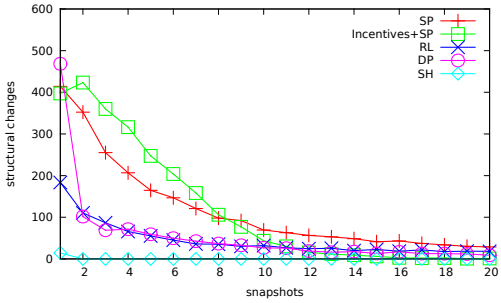


Figure 7.14: Number of structural changes in the system because of the social plasticity of the agents when there are 400 cooperative agents and 600 non-cooperative agents.

non-cooperator agents increased. Therefore, cooperative agents required a higher number of structural changes than with the 'SP' mechanism where agents did not change their behavior and the number of non-cooperators remained constant. The 'DP' strategy generated the highest number of structural changes since there were a high number of links between non-cooperative agents that were broken in the first iterations. The 'SH' strategy generated only a few structural changes since agents preferred the risk of cooperate and this promoted cooperation in the network. This fact decreased the number of links between non-cooperative agents and therefore the number of links that must be broken. The 'RL' strategy generated less structural changes than 'Incentives+SP' and were not enough to promote the cooperation in the network.

7.5 Conclusions

This chapter addresses the problem of emergence of cooperation in scenarios where cooperation is required to achieve a good performance that benefits all of the participants. Specifically, our proposal focuses on the emergence of cooperation in decentralized service discovery scenarios where agents only have a local view of the system and need the cooperation of their neighbors in order to locate other agents

that offer services that they require. Therefore, if selfish agents appear in the system, in the long term, as the number of non-cooperator agents increases, the service discovery process could be seriously compromised. For this reason, it is important to provide mechanisms that facilitate the emergence and maintenance of cooperation. In this chapter, we presented the combination of two mechanisms to facilitate the emergence of cooperation in open service-oriented multi-agent systems where not all the agents have cooperative behavior.

On one hand, agents have social plasticity to change their structural relations based on the degree of cooperation of their neighbors. As the number of times a neighbor refuses to forward a query increases, the probability of changing this relation increases. If an agent decides to change a neighbor, it chooses a neighbor with similar functional features to the previous one. This avoids a high degree of fragmentation of the network when agents that act as hubs (i.e., they have a high number of connections) and non-cooperative agents are isolated by cooperative ones. However, this mechanism may not be appropriate if there are many non-cooperative agents since cooperative agents that are trying to isolate non-cooperative ones could change considerably the structure of the network. Moreover, if many agents are ostracized, the number of potential provider agents that could solve the query is reduced, and, therefore, the probability of success in the service discovery process is reduced. Furthermore, there are scenarios where break links could imply a cost; therefore, not all agent would be able to make use of social plasticity.

On the other hand, we propose the inclusion of incentives in order to influence the behavior of other agents. This mechanism associates a cost or a benefit to each action that an agent can carry out in the service discovery scenario. Forwarding queries have a benefit that depends on the success of the discovery process. Therefore, if the system is populated by a high number of cooperators, forwarding queries is considered to be a beneficial action with high probability. However, as the number of non-cooperators increases, this action becomes reckless and less profitable since the probability of finding an agent that has a neighborhood of non-cooperators increases

and that ensures that the discovery process fails. Incentives are more appropriate for the emergence of cooperation in scenarios where the number of non-cooperators is not greater than the number of cooperators.

In order to deal efficiently with the emergence of cooperation even in scenarios where the number of non-cooperators is higher than the number of cooperators, we have proposed an adaptive combination of social plasticity and incentives. Agents considering local information are able to analyze and change their behavior, influence their neighbors, and decide when it is more appropriate the use social plasticity and incentives mechanisms or when it is enough with the incentives mechanism. With this combination, agents reduce the number of structural changes thereby avoiding the fragmentation of the network and the decrease of potential providers that can be considered during the service discovery process. The structural changes are enough to isolate non-cooperative agents and to increase the effectiveness of incentives in the emergence of cooperation even in scenarios where the majority of agents are non-cooperative. The experiments confirm that this combination of mechanisms promote cooperation in service discovery scenarios with different degrees of cooperation in the population of agents and offer better results than their use separate and than other approaches proposed for promoting cooperation in networks and that are based on game theory or reinforcement learning.

CHAPTER

8

Conclusions

This chapter summarizes the main contributions of this PhD research and identifies future work to extend these contributions. The chapter also presents the list of publications where the main results of the PhD thesis have been presented.

8.1 Contributions

This PhD work presents a model for the management of services in open Service-Oriented Multi-Agent systems. This model integrates different research areas such as Service-Oriented Computing, Multi-Agent Systems, Complex Networks, Semantics, Self-organization and Cooperation. The main contribution of this work is a framework that integrates strategies to facilitate the service management for agents situated in well-structured virtual organizations as well as for agents situated in loosely structured networks. Moreover, the proposed framework allows agents to self-organize

in order to maintain the performance of the system when service demand changes. Due to the openness of Service-Oriented MAS and the requirement of cooperation to carry out the service discovery activity, we have also included strategies that facilitate the promotion and maintenance of cooperation.

On the *State of the Art Revision level*, we have reviewed the literature about structures and search strategies in distributed systems. We have analyzed proposals on the areas of Peer-to-Peer, Service-Oriented Computing, Multi-Agent Systems, and Complex Networks. The analysis is centered on the underlying structure of the systems, how the information is organized, and how the structure conditions the search strategy used to locate the required resources. The works reviewed are organized in three groups: centralized, distributed, and decentralized. Each work is analyzed and discussed taking into account structural features and aspects related to the search process that we consider significant for the development of open distributed systems (i.e., scalability, robustness, adaptability, accuracy, knowledge). Moreover, we have analyzed works related to self-organization and cooperation emergence in distributed systems. Finally, remarks about the desirable features that distributed service-oriented systems should have for the management of resources are presented. These are: (i) scalability and robustness; (ii) capability to deal with a search process relying on local information; (iii) self-organization to environmental changes; (iv) management of different types of information; (v) cooperation promotion; and (vi) integration of functional and non-functional information. This PhD work deals with the majority of these issues.

On the *Formal level*, we have proposed a model that describes the main components that are part of Service-Oriented MAS and their main features. This model describes the system as a set of entities located in an undirected network. The concept of entity integrates individual agents and Virtual Organizations (VOs). Agents are characterized by the organizational role they play. This role determines the type of services offered by an agent that plays that role. Services are defined by their semantic description. The organizational information and the functional information of an agent

is considered to establish structural relations between agents in the network. A Virtual Organization is defined by its goals, the individual agents that are part of it, the roles that are present or required, the structure of the organization, and its functionality. Based on individual agents and VOs, we define a set of actions an agent can perform depending on its situation. There are actions associated to the life-cycle of organizations as well as actions associated to individual agents that deal with decentralized service management or self-organization.

On the *Organization level*, we have presented the THOMAS architecture that integrates services, agents, and organizations. The main roles of this architecture are the *Organization Manager*, which describes the set of services required for the management of the organizational aspects and the life-cycle of the organizations, and the *Service Facilitator*, which describes the set of services required for the management of services provided by entities inside the organizations. Based on this architecture we have designed a new service-oriented execution framework for supporting the development of real VOs. This framework extends previous proposals giving support for VOs, providing functionalities through services that facilitate the interaction among heterogeneous entities in a standardized way, and providing discovery and composition services to facilitate the coordination among entities and the organization adaptation.

On the *Agent level*, we have presented different proposals: (i) how the consideration of social features between agents can create self-organized structures; (ii) how service management can be carried out in a decentralized way; (iii) how agents are able to adapt locally its structure in order to maintain the system performance when the service demand changes; and (iv) how the integration of social plasticity and incentives in the activities of agents can promote the cooperation in service-oriented systems.

The first contribution at Agent level is the *creation of a self-organized structure based on preferences*. These preferences are based on homophily. The concept of homophily is based on the organizational role of the agents and the services they offer.

Agents have more probability to establish links with similar agents than with dissimilar ones. The result of the network structure creation process based on homophily generates networks where agents have the major part of its connections with similar agents and some of them with dissimilar agents. The latter links connect different communities which provides the small-world property of short paths facilitating the service discovery task. Moreover, the exponential degree distribution of this type of networks makes the system less vulnerable to intentional failures than other type of networks such as Scale-Free networks.

The second contribution at Agent level is the proposal of a *service discovery based on local information*. Agents, considering local information about their neighbors degree of connection, and considering the choice homophily between each neighbor and the target agent, are able to decide which of their neighbors has more probabilities to reach the target agent in less steps. Moreover, the experiments show that the consideration of organizational information in the search process improves the results of the service discovery.

The third contribution at Agent level is the proposal of a *reasoning process that allows agents to decide when is more appropriate to make local decisions about their position in the system*. These local decisions facilitate the self-organization of the system when system demand changes. Specifically, agents estimate the utility of their links by analyzing local information about queries received. The weight of each link decays with time if it is not being used. The relationships with neighbors that are not being used are replaced with new structural relations with acquaintances. Acquaintances are established as a result of the service discovery activity. Agents are also able to estimate whether or not they are playing an important role in the system. This estimation is carried out by calculating their structural homophily in the system. Based on their structural homophily, agents decide to remain, leave, or clone themselves in order to adapt the population of the system to the service demand. The use of the self-organization mechanisms maintains the system performance when service demand changes.

Finally, the fourth contribution at Agent level is the inclusion of *mechanisms to promote and maintain cooperation in the system*. The presence of non-cooperative agents in the system can affect seriously to the performance of the service discovery process. Agents in our system have two mechanisms: social plasticity and incentives. The combination of both promote cooperation even in situations where the number of non-cooperators is higher than the cooperators. The inclusion of these two mechanisms facilitate the maintenance of a proper system performance.

Finally, on the *Evaluation level*, we have evaluated each of the contributions through an application example or a set of experiments that consider different scenarios with different configuration setups to analyze the performance of the different proposals.

8.2 Future Lines of Research

Due to the hybrid proposal presented in this PhD work, we can extend this contribution in different directions by advancing research in any of the areas that are covered, such as Service-Oriented Computing, Multi-Agent Systems, Semantics, Complex Networks, Self-Organization, or Cooperation. Here, we mention those that we consider more interesting from our point of view.

Regarding Service-Oriented Computing and Multi-Agent Systems, we consider as the next step for our research the inclusion of decentralized service composition taking as start point the proposal of decentralized service discovery. Moreover, in order to apply the decentralized service discovery in more realistic scenarios, we plan to consider networks of service directories instead of individual agents. This requires the analysis of how directories should be defined, how their local information about services should be organized internally, how the network should be created in a decentralized way, and which criteria is appropriate to facilitate the service location. Another future work related to the decentralized service discovery is exploring alternative solutions instead of stopping the service discovery process when the first solution is found. If alternative solutions are considered, we plan to include non-

functional parameters related to the quality of the services to rank the solutions.

Regarding the area of Self-Organization, there are several possibilities that can be still investigated. One of them is the consideration of costs associated to the different adaptation actions such as rewiring or cloning. Moreover, in this thesis we have assumed that agents have an unbounded number of resources, therefore, they always have enough computational resources to attend and analyze all the queries received during the service discovery. However, real application scenarios can impose these constraints to the resources of agents.

Regarding the area of Cooperation, we want to include several considerations that would create a more complex scenarios for the cooperation emergence. Some of these considerations are: bounded economical resources in the system, associate costs to social plasticity, or shared costs of social plasticity between agents that are located in the neighborhood. Another issue that we want to introduce in a future work is reputation mechanisms in order to avoid the entry or reconnection of non-cooperative agents.

Furthermore, depending on the application domain, the preferences over values of each agent in self-organization and cooperation mechanisms can change with the curse of the time. In this thesis we have assumed that all agents have the same values for the parameters in the mechanisms and that their values do not change. This is not realistic, and therefore, requires adaption methods that allow agents to reconfigure its self-organization and cooperation parameters dynamically.

8.3 Related Publications

This section presents the publications associated with the PhD thesis that have been published to date. The first list includes publications in SCI Journals. The second list includes those publications that are indexed in a ranking. Two rankings are considered: the CORE and the Computer Science Conference Ranking. The third list

includes publications from workshops that are not indexed.

8.3.1 Indexed Journals

- E. Del Val, M. Rebollo and V. Botti. An Overview of Search Strategies in Distributed Environments. The Knowledge Engineering Review pp. In Press. (2012) **JCR 1.257**

- E. Del Val, M. Rebollo and V. Botti. Enhancing Decentralized Service Discovery in Open Service-Oriented Multi-Agent Systems. Journal of Autonomous Agents and Multi-Agent Systems pp. 1-30. (2012) DOI: 10.1007/s10458-012-9210-0 **JCR 1.213 Q2**

- E. Del Val, M. Rebollo and V. Botti. Promoting Cooperation in Service-Oriented MAS through Social Plasticity and Incentives. Journal of Systems and Software. Vol 86, 2 (February 2013) p. 520-537 **JCR 0.836 Q2**

8.3.2 Indexed Conferences

- E. Del Val, M. Rebollo and V. Botti. Emergence of Cooperation through Structural Changes and Incentives in Service-Oriented MAS. Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems pp. 1355-1356. (2012) **CORE A**

- E. Del Val, M. Vassirani, M. Rebollo and A. Fernández. Enhancing Decentralized Service Discovery through Structural Self-Organization. Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems pp. 1429-1430. (2012) **CORE A**

- E. Del Val, M. Rebollo and V. Botti. Self-Organized Service Management in Social Systems. Proceedings of the 45th Hawaii International Conference on System Sciences pp. 810-817. (2012) **Best paper nomination CORE A**

- E. Del Val, M. Rebollo and V. Botti. Decentralized Service Management based on Homophily for Self-Adaptive SOMAS. Proceedings of the 8th IEEE International Conference on Services Computing pp. 755-756. (2011) **CORE A**

- E. Del Val. Decentralized Semantic Service Discovery based on Homophily for Self-Adaptive Service-Oriented MAS. Proceedings of the 10th Int. Conf. on Autonomous Agents and Multiagent Systems pp. 1347-1348. (2011) **CORE A**

- E. Del Val, M. Rebollo and V. Botti. Introducing homophily to improve semantic service search in a self-adaptive system (Extended Abstract). Proceedings of the 10th Int. Conf. on Autonomous Agents and Multiagent Systems pp. 1242-1243. (2011) **CORE A**

- E. Del Val, N. Criado, C. Carrascosa, V. Julian, M. Rebollo, E. Argente and V. Botti. THOMAS: A Service-Oriented Framework For Virtual Organizations. Proceedings of the 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010) pp. 1631-1632. (2010) **CORE A**

- E. Del Val, M. Rebollo and V. Botti. Decentralized Semantic Service Discovery in MAS. Proceedings of the 8th European Workshop on Multi-Agent Systems pp. 1-12. (2010) **CORE C**

- E. Del Val, M. Navarro, V. Julian and M. Rebollo. Managing Real-Time Web Services Through Agents. Proceedings of the International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS) Vol. 71 pp. 61-68. (2010). **Computer Science Conference Ranking: 0.56 (position 51 / 701)**

- E. Del Val, N. Criado, M. Rebollo and E. Argente. Normative Time-Bounded Service Logic. Proceedings of the 7th European Workshop on Multi-Agent Systems (EUMAS'09) pp. 1-13. (2009) **CORE C**

- N. Criado, E. Del Val, M. Rebollo and E. Argente. A Logic for Normative Time-Bounded Services. Proceedings of the Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA) pp. 645-654. (2009) **Computer Science Conference Ranking: 0.55 (position 54 / 701)**

- E. Del Val, M. Navarro, V. Julian and M. Rebollo. Ensuring time in service composition. Proceedings of the IEEE Congress on Services (SERVICES 2009) Vol. 1 pp. 376-383. (2009) **CORE B**

- E. Del Val, M. Navarro, V. Julian and M. Rebollo. A framework to guarantee time-bounded composed services. Proceedings of the IEEE / WIC / ACM International Conferences on Web Intelligence pp. 434-437. (2009) **Computer Science Conference Ranking: 0.82 (position 16 / 701)**

- M. Navarro, E. Del Val, M. Rebollo and V. Julian. Composing and Ensuring Time-Bounded Agent Services. Proceedings of the International Work-Conference on Artificial Neural Networks (IWANN'09) Vol. 1 N. 5517 pp. 553-560. (2009) **Computer Science Conference Ranking: 0.55 (position 55 / 701)**

- E. Del Val, N. Criado, M. Rebollo, E. Argente and V. Julian. Service-Oriented Framework for Virtual Organizations. Proceedings of the International Conference on Artificial Intelligence (ICAI) Vol. 1 pp. 108-114. (2009) **Computer Science Conference Ranking: 0.8 (positionn 18 / 701)**

- E. Del Val and M. Rebollo. A SURVEY ON WEB SERVICE DISCOVERING AND COMPOSITION. Proceedings of the International Conference on Web Information Systems and Technologies (Webist) Vol. I pp. 135-142. (2008) **CORE C**

- E. Del Val and M. Rebollo. Service Discovery and Composition in Multiagent Systems. Proceedings of the 5th European Workshop On Multi-Agent Systems (EUMAS 2007) pp. 197-212. (2007) **CORE C**

8.3.3 Other Conferences

- E. Del Val, M. Rebollo and V. Botti. Composition of Temporal Bounded Services in Open MAS. International Symposium on Distributed Computing, Artificial Intelligence, Bioinformátics, Soft Computing, and Ambient Assisted Living Vol. 5518 pp. 146-154. (2009)

- M. Navarro, E. Del Val, M. Rebollo and V. Julian. Agent Negotiation Protocols in Time-Bounded Service Composition. Intelligent Data Engineering and Automated Learning N. 5788 pp. 527-534. (2009)

8.3.4 Workshops

- E. Del Val, M. Rebollo and V. Botti. Strategies for cooperation emergence in distributed service discovery. Proceedings of the Workshop on Conflict Resolution in Decision Making 2013. In press (2013)

- E. Del Val, M. Rebollo and V. Botti. Semantic Service Discovery in MAS Using Social Networks. Proceedings of the Workshop on Agreement Technologies 2010 Vol. 657 pp. 23-34. (2010)

- E. Del Val, M. Rebollo and V. Botti. Decentralized Semantic Service Discovery in Preferential Attachment Networks. Proceedings of the Workshop on Agent-based Technologies and applications for enterprise interOPerability (ATOP 2010) pp. 71-82. (2010).

8.3.5 Book Chapters

- E. Del Val, M. Rebollo and V. Botti. Decentralized Semantic Service Discovery in Preferential Attachment Networks. Agent-Based Technologies and Applications for Enterprise Interoperability. Lecture Notes in Business Information Processing, 2012, Volume 98, Part 2, 130-150, DOI: 10.1007/9783642285639_8

- Bonatti, P., del Val, E., Fernandez, A., Florea, A.M., Jezic, G., Paprzycki, M., Polleres, A., Rebollo, M., Vouros, G., Zimmermann, A. White Paper: Semantics in Agreement Technologies. Distributed Computing, Artificial Intelligence, Bioinformátics, Soft Computing, and Ambient Assisted Living (2010)

Bibliography

[1] *Self-organizing Software: From Natural to Artificial Adaptation*. Self-Organising Software From Natural to Artificial Adaptation - Natural Computing Series. Springer, first edition, 2011.

[2] S. Abdallah and V. Lesser. Multiagent reinforcement learning and self-organization in a network of agents. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 172–179. IFAAMAS, May 2007.

[3] Adamic. Zipf's law and the internet. *Glottometrics*, 3:143–150, 2002.

[4] L. A. Adamic and E. Adar. How to search a social network. *Social Networks*, 27:2005, 2005.

[5] E. Argente, V. Botti, C. Carrascosa, A. Giret, V. Julian, and M. Rebollo. An Abstract Architecture for Virtual Organizations: The THOMAS approach. *Knowledge and Information Systems*, pages 1–35, 2011.

[6] R. M. Axelrod. *The evolution of cooperation*. Basic Books, New York, 1984.

[7] D. Bachlechner, K. Siorpaes, D. Fensel, and I. Toma. Web service discovery – a reality check. In *Proceedings of the 3rd European Semantic Web Conference*, 2006.

[8] J. Bailey. Fast discovery of interesting collections of web services. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 152–160. IEEE Computer Society, 2006.

[9] U. Basters and M. Klusch. Rs2d: Fast adaptive search for semantic web services in unstructured p2p networks. In *Proceedings of the International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 87–100. Springer, 2006.

[10] D. Ben-Ami and O. Shehory. A comparative evaluation of agent location mechanisms in large scale mas. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '05, pages 339–346, New York, NY, USA, 2005. ACM.

[11] D. Bianchini, V. D. Antonellis, and M. Melchiori. Service-based semantic search in p2p systems. volume 0, pages 7–16, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[12] H. Bisgin, N. Agarwal, and X. Xu. Investigating homophily in online social networks. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '10, pages 533–536, Washington, DC, USA, 2010. IEEE Computer Society.

[13] B. Biskupski, J. Dowling, and J. Sacha. Properties and mechanisms of self-organizing manet and p2p systems. *ACM Trans. Auton. Adapt. Syst.*, 2, 2007.

[14] N. Bisnik and A. Abouzeid. Modeling and analysis of random walk search algorithms in p2p networks. In *Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 95–103. IEEE Computer Society, 2005.

[15] A. Blanc, Y.-K. Liu, and A. Vahdat. Designing incentives for peer-to-peer routing. In *Proceedings of the 24th Annual Joint Conference of the IEEE*

Computer and Communications Societies., volume 1, pages 374 – 385, march 2005.

[16] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang. Complex networks: Structure and dynamics. *Physics Reports*, 424(4–5):175 – 308, 2006.

[17] G. Boella and L. V. D. Torre. Coordination and organization: Definitions, examples and future research directions. In *Proceedings of the First International Workshop on Coordination and Organisation*, pages 3–20, 2005.

[18] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250, 2002.

[19] F. M. T. Brazier, J. O. Kephart, H. V. D. Parunak, and M. N. Huhns. Agents and service-oriented computing for autonomic computing: A research agenda. *IEEE Internet Computing*, 13, 2009.

[20] A. Brogi, S. Corfini, J. Aldana, and I. Navas. Automated discovery of compositions of services described with separate ontologies. pages 509–514. A. Dan and W. Lamersdorf, 2006.

[21] S. Bromuri, V. Urovi, M. Morge, K. Stathis, and F. Toni. A multi-agent system for service discovery, selection and negotiation. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1395–1396. IFAAMAS, 2009.

[22] C. Cáceres, A. Fernández, S. Ossowski, and M. Vasirani. Role-based service description and discovery. In *In Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2006.

[23] J. Cao, Y. Yao, X. Zheng, and B. Liu. Semantic-based self-organizing mechanism for service registry and discovery. In *Proceedings of the 14th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 345 –3350, 2010.

[24] M. Carman, L. Serafini, and P. Traverso. Web Service Composition as Planning. In *Proceedings of the Workshop on Planning for Web Services*, 2003.

[25] G. D. Caro, F. Ducatelle, and L. M. Gambardella. Anthocnet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *European Transactions On Telecommunications*, 16:443–455, 2005.

[26] D. Centola, J. GonzalezAvella, V. Eguiluz, and M. SanMiguel. Homophily, cultural drift, and the co-evolution of cultural groups. *Journal of Conflict Resolution*, 2007.

[27] S. Chaari, Y. Badr, and F. Biennier. Enhancing web service selection by qos-based ontology and ws-policy. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 2426–2431. ACM, 2008.

[28] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 407–418. ACM, 2003.

[29] B.-G. Chun, R. Fonseca, I. Stoica, and J. Kubiatowicz. Characterizing selfishly constructed overlay routing networks. In *INFOCOM*, 2004.

[30] T. Condie, S. D. Kamvar, and H. Garcia-Molina. Adaptive peer-to-peer topologies. volume 0, pages 53–62, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[31] Z. Cong and A. Fernandez. Behavioral matchmaking of semantic web services. In *Proceedings of the 4th International Joint Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web (SMR2)*, volume 667, pages 131–140, 2010.

[32] A. Crespo and H. Garcia-Molina. Routing Indices For Peer-to-Peer Systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, page 23. IEEE Computer Society, 2002.

[33] N. Criado, E. Argente, and V. Botti. THOMAS: An Agent Platform For Supporting Normative Multi-Agent Systems. *Journal of Logic and Computation*, 2011.

[34] S. Currarini and F. Vega-Redondo. Search and homophily in social networks. *World*, (24):1–32, 2010.

[35] E. Del Val and M. Rebollo. Service Discovery and Composition in Multiagent Systems. In *Proceedings of Fifth European Workshop On Multi-Agent Systems (EUMAS 2007)*, pages 197–212. Association Tunisienne D'Intelligence Artificielle, 2007.

[36] E. Del Val, M. Rebollo, and V. Botti. Introducing homophily to improve semantic service search in a self-adaptive system. In *Proceedings of the 10th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1241–1242, 2011.

[37] E. Del Val, M. Rebollo, and V. Botti. An Overview of Search Strategies in Distributed Environments. *The Knowledge Engineering Review*, 2012.

[38] E. Del Val, M. Rebollo, and V. Botti. Emergence of Cooperation through Structural Changes and Incentives in Service-Oriented MAS. In *11th International Conference on Autonomous Agents and Multiagent Systems*, pages 1355–1356, 2012.

[39] G. Di Marzo Serugendo, M. P. Gleizes, and A. Karageorgos. Self-organization in multi-agent systems. *Knowl. Eng. Rev.*, pages 165–189, 2005.

[40] F. Dignum, V. Dignum, J. Thangarajah, L. Padgham, and M. Winikoff. Open Agent Systems. In *Proceedings of the International Workshop on Agent Oriented Software Engineering*, pages 73–87, 2007.

[41] L. Donetti and M. Munoz. Detecting network communities: a new systematic and efficient algorithm. *Journal of Statistical Mechanics: Theory and Experiment*, (10), 2004.

[42] J. E. Doran, S. Franklin, N. R. Jennings, and T. J. Norman. On cooperation in multi-agent systems. *The Knowledge Engineering Review*, 12:309–314, 1997.

[43] S. Dorogovtsev and J. Mendes. *Evolution of Networks*. 2003.

[44] V. M. Eguluz, M. G. Zimmermann, C. J. Cela-Conde, and M. San Miguel. Cooperation and emergence of role differentiation in the dynamics of social networks. *American Journal of Sociology*, 110:977, 2005.

[45] E. Einhorn and A. Mitschele-Thiel. Rlte: Reinforcement learning for traffic-engineering. In *Proceedings of the 2nd international conference on Autonomous Infrastructure, Management and Security: Resilient Networks and Services*, AIMS '08, pages 120–133, Berlin, Heidelberg, 2008. Springer-Verlag.

[46] J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations: An organizational view of multi-agent systems. In *Proceedings of the 4th International Workshop on Agent-Oriented Software Engineering*, volume 2935 of *Lecture Notes in Computer Science*, pages 214–230. Springer, 2003.

[47] A. Fernandez, S. Ossowski, and M. Vasirani. General architecture. In *CASCOM: Intelligent Service Coordination in the Semantic Web*, Whitestein Series in Software Agent Technologies and Autonomic Computing, pages 143–160. 2008.

[48] A. Forestiero, C. Mastroianni, and M. Meo. Self-chord: A bio-inspired algorithm for structured p2p systems. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 44–51, 2009.

[49] M. Fox and D. Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124, 2003.

[50] P. Fu, S. Liu, H. Yang, and L. Gu. Matching algorithm of web services based on semantic distance. In *Proceedings of 2009 International Workshop on Information Security and Application*, 2009.

[51] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks: Algorithms and evaluation. *Performance Evaluation*, 63(3):241–263, March 2006.

[52] N. Griffiths and M. Luck. Changing neighbours: improving tag-based cooperation. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, pages 249–256. IFAAMAS, 2010.

[53] B. Gu and S. Jarvenpaa. Are contributions to p2p technical forums private or public goods? - an empirical investigation. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*, 2003.

[54] Y. Guanfeng Liu and M. A. Orgun. Optimal social trust path selection in complex social networks. *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, 2010.

[55] P. K. Gummadi, S. Saroiu, and S. D. Gribble. A measurement study of napster and gnutella as examples of peer-to-peer file sharing systems. *SIGCOMM Comput. Commun. Rev.*, 32:82–82, 2002.

[56] D. Hales and B. Edmonds. Applying a socially inspired technique (tags) to improve cooperation in p2p networks. *IEEE Trans. Systems, Man, and Cybernetics—Part A: Systems and Humans*, pages 385–395, 2005.

[57] G. Hardin. The tragedy of the commons. *Science*, (162):1243–1248, 1968.

[58] C. Hauert, A. Traulsen, H. Brandt, M. A. Nowak, and K. Sigmund. Via Freedom to Coercion: The Emergence of Costly Punishment. *Science*, 316(5833):1905–1907, 2007.

[59] Q. He, J. Yan, Y. Yang, R. Kowalczyk, and H. Jin. Chord4s: A p2p-based decentralised service discovery approach. In *Proceedings of the IEEE International Conference on Services Computing*, volume 1, pages 221 –2228, 2008.

[60] L.-M. Hofmann, N. Chakraborty, and K. Sycara. The evolution of cooperation in self-interested agent societies: a critical study. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, pages 685–692. IFAAMAS, 2011.

[61] B. A. Huberman and L. A. Adamic. The nature of markets in the www. Technical report, 1999.

[62] D. Hughes, J. Walkerdine, and G. Coulson. A survey of peer-to-peer architectures for service oriented computing. *Network*, pages 1–19, 2010.

[63] M. N. Huhns. Agents as web services. *IEEE Internet Computing*, pages 93–95, 2002.

[64] M. N. Huhns, M. P. Singh, M. Burstein, K. Decker, E. Durfee, T. Finin, L. Gasser, H. Goradia, N. Jennings, K. Lakkaraju, H. Nakashima, V. Parunak, J. S. Rosenschein, A. Ruvinsky, G. Sukthankar, S. Swarup, K. Sycara, M. Tambe, T. Wagner, and L. Zavala. Research directions for service-oriented multiagent systems. *IEEE Internet Computing*, 9:65–70, 2005.

[65] I.Foster, C.Kesselman, and S.Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl*, 2001.

[66] E. Kaddoum, C. Raibulet, J.-P. Georgé, G. Picard, and M.-P. Gleizes. Criteria for the evaluation of self-* systems. *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and SelfManaging Systems*, pages 29–38, 2010.

[67] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. In *Proceedings of the 11th international*

conference on Information and knowledge management, pages 300–307, New York, NY, USA, 2002. ACM.

[68] S. Kamboj and K. S. Decker. Organizational self-design in semi-dynamic environments. In *Proceedings of the 6th International Conference on Autonomous Agents and Multiagent Systems*, pages 202:1–202:8. ACM, 2007.

[69] C. Kamhoua, N. Pissinou, and K. Makki. Game theoretic modeling and evolution of trust in autonomous multi-hop networks: Application to network security and privacy. In *Proceedings of the 2011 IEEE International Conference on Communications (ICC)*, pages 1 –16, june 2011.

[70] J. Kleinberg. Complex networks and decentralized search algorithms. In *Proceedings of the International Congress of Mathematicians (ICM)*, 2006.

[71] J. M. Kleinberg. Navigation in a small world. *Nature*, 406:845, 2000.

[72] M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with owls-mx. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 915–922. ACM, 2006.

[73] M. Klusch, B. Fries, and K. Sycara. Owls-mx: A hybrid semantic web service matchmaker for owl-s services. *Web Semantics Science Services and Agents on the World Wide Web*, 7(2):121–133, 2009.

[74] M. Klusch and A. Gerber. Semantic web service composition planning with owls-xplan. In *Proceedings of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web*, pages 55–62, 2005.

[75] R. Kota, N. Gibbins, and N. Jennings. Self-organising agent organisations. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, 2009.

[76] M. Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theor. Comput. Sci.*, 407:458–473, November 2008.

[77] P. Lazarsfeld. Friendship as a social process: A substantive and methodological analysis. *Freedom and Control in Modern Society*, 1954.

[78] J. Liang, R. Kumar, and K. Ross. Understanding kazaa, 2004.

[79] E. Lieberman, C. Hauert, and M. A. Nowak. Evolutionary dynamics on graphs. *Nature*, 433(7023):312–316, Jan. 2005.

[80] W. Lin, H. Zhao, and K. Liu. Incentive cooperation strategies for peer-to-peer live multimedia streaming social networks. *Multimedia, IEEE Transactions on*, 11(3):396 –3412, april 2009.

[81] L. Liu and H. Schmeck. Enabling self-organising service level management with automated negotiation. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, WI-IAT '10, pages 42–45. IEEE Computer Society, 2010.

[82] A. L. Lopes and L. M. Botelho. Improving multi-agent based resource coordination in peer-to-peer networks. *Journal of Networks*, 3:38–47, 2008.

[83] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7:72–93, 2005.

[84] M. Luck and P. McBurney. Computing as interaction: Agent and agreement technologies. 2008.

[85] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th international conference on Supercomputing*, ICS '02, pages 84–95, New York, NY, USA, 2002. ACM.

[86] G. S. Manku, M. Bawa, P. Raghavan, and V. Inc. Symphony: Distributed hashing in a small world. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, pages 127–140, 2003.

[87] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. *Proceedings of the 1st International Workshop on Peer-to Peer Systems (IPTPS02)*, 2002.

[88] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16:46–53, March 2001.

[89] M. McPherson, L. Smith-Lovin, and J. Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 2001.

[90] E. Meshkova, J. Riihijärvi, M. Petrova, and P. Mähönen. A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 52:2097–2128, 2008.

[91] E. Michlmayr. Ant algorithms for search in unstructured peer-to-peer networks. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*, 2006.

[92] S. Mokhtar, A. Kaul, N. Georgantas, and V. Issarny. Towards efficient matching of semantic web service capabilities. In *Proceedings of WSMaTe 2006*, 2006.

[93] M. Moore and T. Suda. A decentralized and self-organizing discovery mechanism. In *Proceedings Of the 1st Annual Symposium on Autonomous Intelligent Networks and Systems*, 2002.

[94] S. Mullender and P. Vitanyi. Distributed match-making. *Algorithmica*, 3:367–391, 1988.

[95] M. E. J. Newman. Assortative mixing in networks. *Phys.Rev.Lett.*, 89:208701, 2002.

[96] M. E. J. Newman. Mixing patterns in networks. *Phys. Rev. E*, (67), 2003.

[97] M. A. Nowak. Five Rules for the Evolution of Cooperation. *Science*, 314(5805):1560–1563, 2006.

[98] M. A. Nowak and K. Sigmund. Evolution of indirect reciprocity by image scoring. *Nature*, 393(6685):573–577, 1998.

[99] E. Ogston and S. Vassiliadis. Local distributed agent matchmaking. In *Proceedings of the 9th International Conference on Cooperative Information Systems*, 2001.

[100] E. Ogston and S. Vassiliadis. Matchmaking among minimal agents without a facilitator. In *Proceedings of the 5th International Conference on Autonomous Agents*, pages 608–615, 2001.

[101] H. Ohtsuki, C. Hauert, E. Lieberman, and M. A. Nowak. A simple rule for the evolution of cooperation on graphs and social networks. *Nature*, 441(7092):502–505, 2006.

[102] S. Ossowski, V. Julian, J. Bajo, H. Billhardt, V. Botti, and J. Corchado. Open mas for real world applications: an abstract architecture proposal. In *Proceedings of the Conference of the Spanish Association for Artificial Intelligence (CAEPIA)*, pages Volumen II, 151–160, 2007.

[103] A. Ouksel, Y. Babad, and T. Tesch. Matchmaking software agents in b2b markets. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, 2004.

[104] M. Paolucci. Semantic matching of web services capabilities. In *Proceedings of the 1st International Semantic Web Conference*, volume 2342 of *Lecture Notes in Computer Science*, pages 333–347, 2002.

[105] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40:38–45, 2007.

[106] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented computing: A research roadmap. *International Journal of Cooperative Information Systems*, 17(02):223+, 2008.

[107] J. Peer. Web service composition as ai planning – a survey. *Language*, (March), 2005.

[108] C. A. Perryea and S. Chung. Community-based service discovery. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '06, pages 903–906, Washington, DC, USA, 2006. IEEE Computer Society.

[109] L. Peshkin and V. Savova. Reinforcement learning for adaptive routing. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 1825 –11830, 2002.

[110] G. Pirro, P. Trunfio, D. Talia, P. Missier, and C. Goble. Ergot: A semantic-based system for service discovery in distributed infrastructures. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 263 –2272, 2010.

[111] S. Prabhu. Towards distributed dynamic web service composition. In *Proceedings of the 8th International Symposium on Autonomous Decentralized Systems*, pages 25–32, Washington, DC, USA, 2007. IEEE Computer Society.

[112] J. M. Pujol, J. Delgado, R. Sangüesa, and A. Flache. The role of clustering on the emergence of efficient social conventions. In *Proceedings of the 19th international joint conference on Artificial intelligence*, pages 965–970, 2005.

[113] P. Raftopoulou and E. G. M. Petrakis. icluster: a self-organizing overlay network for p2p information retrieval. In *Proceedings of the 30th European conference on Advances in information retrieval*, pages 65–76, Berlin, Heidelberg, 2008. Springer-Verlag.

[114] C. Raibulet and L. Masciadri. Evaluation of dynamic adaptivity through metrics: an achievable target? In *Proceedings of the European Conference on Software Architecture*, pages 341 –3344, 2009.

[115] J. Rao and X. Su. A survey of automated web service composition methods. *Semantic Web Services and Web Process Composition*, 3387/2005:43–54, 2005.

[116] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, New York, NY, USA, 2001. ACM.

[117] J. Risson and T. Moors. Survey of research towards robust peer-to-peer networks: search methods. *Computer Networks*, 50:3485–3521, December 2006.

[118] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, pages 329–350. Springer-Verlag, 2001.

[119] N. Salazar, J. A. Rodríguez-Aguilar, J. L. Arcos, A. Peleteiro-Ramallo, and J. C. Burguillo-Rial. Emerging cooperation on complex networks. In *Proceedings of the 10th Int. Conf. on Autonomous Agents and Multiagent Systems*, pages 669–676, 2011.

[120] F. C. Santos, M. D. Santos, and J. M. Pacheco. Social diversity promotes the emergence of cooperation in public goods games. *Nature*, 454(7201):213–216, 2008.

[121] J. Shneidman and D. C. Parkes. Rationality and self-interest in peer to peer networks. In *Proceedings of the 2nd Int. Workshop on Peer-to-Peer Systems (IPTPS'03)*, 2003.

[122] P. Shvaiko and J. Euzenat. Ten challenges for ontology matching. In *Proceedings of the OTM 2008 Confederated International Conferences*, volume 5332 of *Lecture Notes in Computer Science*, pages 1164–1182. Springer Berlin / Heidelberg, 2008.

[123] C. Sierra, J. Thangarajah, L. Padgham, and M. Winikoff. Designing institutional multi-agent systems. In *Agent-Oriented Software Engineering VII*, volume 4405 of *Lecture Notes in Computer Science*, pages 84–103. Springer-Verlag, 2007.

[124] K. Sigdel, K. Bertels, B. Pourebrahimi, S. Vassiliadis, and L. Shuai. A framework for adaptive matchmaking in distributed computing. In *Proceedings of GRID Workshop*, 2005.

[125] Sigmund, C. Hauert, and M. Nowak. Reward and punishment. *P Natl Acad Sci USA*, (19):10757–10762, 2001.

[126] K. Sigmund. Punish or perish? retaliation and collaboration among humans. *Trends in Ecology and Evolution*, 22(11):593 – 600, 2007.

[127] K. Sigmund. Sympathy and similarity: The evolutionary dynamics of cooperation. *Proceedings of the National Academy of Sciences*, 106(21):8405–8406, 2009.

[128] Ö. Simsek and D. Jensen. Decentralized search in networks using homophily and degree disparity. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 304–310, 2005.

[129] K. Sivashanmugam, K. Verma, and A. Sheth. Discovery of web services in a federated registry environment. volume 0, page 270, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[130] D. Skoutas, D. Sacharidis, V. Kantere, and T. Sellis. Efficient semantic web service discovery in centralized and p2p environments. In *Proceedings of the International Semantic Web Confgerence*, volume 5318, pages 583–598, 2008.

[131] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. *Computer Communication Review*, 31(4):149–160, 2001.

[132] D. Strang and M. W. Macy. In search of excellence: Fads, success stories, and adaptive emulation. *American Journal of Sociology*, 107:107, 2001.

[133] Q. Sun and H. Garcia-Molina. Slic: A selfish link-based incentive mechanism for unstructured peer-to-peer networks. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 506–515, Washington, DC, USA, 2004. IEEE Computer Society.

[134] D. Tsoumakos and N. Roussopoulos. Adaptive probabilistic search for peer-to-peer networks. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, pages 102–109, 2003.

[135] Y. Upadrashta, J. Vassileva, and W. Grassmann. Social networks in peer-to-peer systems. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 2005.

[136] K. Vanthournout, G. Deconinck, and R. Belmans. A taxonomy for resource discovery. *Personal Ubiquitous Comput.*, 9:81–89, 2005.

[137] D. Villatoro, J. Sabater-Mir, and S. Sen. Social instruments for robust convention emergence. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 420–425, 2011.

[138] L. Wang. Sofa: An expert-driven, self-organization peer-to-peer semantic communities for network resource management. *Expert Syst. Appl.*, 38(1):94–105, Jan. 2011.

[139] D. Watts, P. Dodds, and M. Newman. Identity and search in social networks. *Science*, 296(5571):1302 – 1305, 2002.

[140] D. J. Watts. The "New" Science of Networks. *Annu. Rev. Sociol.*, 30:243–270, 2004.

[141] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[142] Y. Wei and M. B. Blake. Service-oriented computing and cloud computing: Challenges and opportunities. *IEEE Internet Computing*, 14:72–75, 2010.

[143] D. Weyns and M. Georgeff. Self-adaptation using multiagent systems. *IEEE Software*, 27(1):86 –891, 2010.

[144] S. Xiao and G. Xiao. On degree-based decentralized search in complex networks. *CoRR*, 2006.

[145] B. Yang and H. Garcia-Molina. Efficient search in peer-to-peer networks. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, 2002.

[146] H. Zhang, W. B. Croft, B. Levine, and V. Lesser. A multi-agent approach for peer-to-peer based information retrieval system. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, pages 456–463. IEEE Computer Society, 2004.

[147] M. Zhong. Popularity-biased random walks for peer-to-peer search under the square-root principle. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.