Combination of Self-Organization Mechanisms to Enhance Service Discovery in Open Systems

E. del Val, M. Rebollo, V. Botti
{edelval,mrebollo,vbotti@dsic.upv.es}

Universitat Politècnica de València
València, Spain

Abstract

Decentralized systems have emerged as an alternative to centralized approaches for dealing with dynamic requirements in new business models. These systems should provide mechanisms that contribute to flexibility and facilitate adaptation to changes in the environment. In this paper, we present two self-organization mechanisms for a decentralized service discovery system in order to improve its performance. These mechanisms are based on local actions of agents that only consider local information about queries they forward during the discovery process. The self-organization actions are chosen by each agent individually when the agent considers them to be appropriate. The actions are: remaining in the system, leaving the system, cloning, and changing structural relations with other agents. We have evaluated each self-organization mechanism separately but also the combination of the two as the environmental conditions in the service demand change. The results show that the proposed self-organization mechanisms considerably improve the performance of the service discovery system.

Keywords: Self-organization, adaptation, service discovery

1. Introduction

Nowadays, there is a trend towards large-scale, complex, and highly-dynamic systems in order to deal with new business models and requirements. Peer-to-peer technologies (P2P) [37], Service-Oriented Computing (SOC) [33], or Cloud Computing [7] are considered to be suitable technologies to support these new models where there is a high number of entities offering services that change frequently

and look for other entities to collaborate with in order to obtain a resource or to deal with a complex goal.

To facilitate collaboration between entities, systems should provide mechanisms to manage 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. However, this is not an easy task in open and dynamic environments where there are frequent changes in the available resources and global information is not always available. Under these circumstances, the management and location of available resources become more difficult. The field of Complex Networks has emerged as an alternative to be able to deal with decentralized service management in a flexible and adaptive way [5]. Some of the models proposed in this area provide structures that allow the location of resources in a few steps taking only local information into account. One of the properties present in some of these structures is homophily [44, 39, 23]. The idea behind this social concept is that individuals tend to interact and establish links with similar individuals along a set of social dimensions. Therefore, in a structure that is based on homophily, an individual has a higher probability of being connected to a more similar individual than to a dissimilar one. This criterion creates structures that facilitate the location task and could be considered to be a self-organizing principle for generating searchable structures.

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). 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 paper, we present a decentralized service management system for service-oriented environments. Specifically, we propose the use of Multi-Agent Systems and Service-Oriented Computing as appropriate paradigms for building these systems [14]. Agents in the proposed system offer their functionality through services and have a collaborative behavior that facilitates the decentralized location of services using only local knowledge. Agents are located in a network where structural relations between agents are based on the self-organizing concept of homophily. The structural relations determine the interactions between the agents, their local knowledge, and, therefore, the performance of the service discovery process. The number of agents and the structural relations between agents do

no remain static in open systems. In this way, 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 paper is structured as follows: in section 2, we present an overview of works that includes distributed search and self-organization strategies in distributed environments. In section 3, an example of a service discovery scenario is presented. In section 4, we describe the formalization of our proposal for the self-organization of the decentralized discovery system. In section 5, the self-organization actions that agents can execute during the service discovery are described. In section 6, we present a set of experiments to validate the proposed model and the self-organization mechanisms in different scenarios. Finally, in section 7, conclusions and final remarks are presented.

2. Related Work

Large-scale, open, and highly-dynamic systems are populated by entities that have to deal with complex tasks and need services provided by other entities in order to fulfill their goals. Therefore, these systems should provide mechanisms to manage the information about the available services in the system and to determine which entities provide them. Moreover, in order to deal with changes in the requirements or in the environment conditions, these systems should provide self-organization functionalities. *Self-organization* is considered to be the mechanism or the process that enables a system to arrange its organization at run-time, without explicit external commands [13]. 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 [24]. 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 [45]. 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. The adaptation process should be carried out locally, based on the local interactions of each entity; (ii) the system should be able to evolve; (iii) the entities of the system should be able to deal with uncertainty in order to make decisions. In this context, researchers have proposed mechanisms that deal with distributed service discovery and self-organization in several ways.

In distributed approaches, the responsibility of resource management relies on a set of specific entities to provide scalability and robustness. In P2P systems, structures based on *super-peers* [36] and *Distributed Hash Tables* (DHT) [40, 30] 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 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. There are some approaches based on super-peers that deal with this problem introducing semantic information. In these systems, peers with similar content connect to the same super-peer and sophisticated routing strategies based on the metadata schema, attributes and ontologies are used [31].

There are other works based on 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 systems use *blind* or *informed* algorithms for locating 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 [47]. In general, flooding algorithms overload the system with the traffic generated during the search process. However, there are approaches that try to improve the efficiency of flooding strategies using self-organization mechanisms. Cooper *et al.* [8] present a work where peers are initially randomly connected and when their links are overloaded, agents can disconnect them. The peers that are disconnected will then reconnect to other peers. *Random walks* have been presented as an alternative search algorithms to flooding ones [42]. A random walk algorithm selects a set of the neighbors to forward the message. Each message follows its own path and is called a walker. The disadvantage of these types of algorithms is that the percentage of success varies depending on the network topology [17], the popularity of the resource, the number of walkers, and the Time To Live.

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.* [9]. 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 is 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. Tempich *et al.* [41] present an approach that also uses techniques of routing indices. Peers observe which queries are successfully solved by other peers and remember these peers in future query routing decisions. To improve the precision in the search results, semantic descriptions are used for the data sources. There are other approaches that use algorithms for information sharing that are based on the relationship between pieces of information to enable efficient information sharing [46]. The pieces of information are obtained from previous sent messages between agents. Therefore, when an agent has a piece of information, based on previous messages, it can choose which of its neighbors is most likely to need the information or know who has it. The main drawback is how to calculate the relationship between pieces of information since it is highly domain dependent. Yu and Singh [49] propose the use of

a referral system in order to guide the search process. Each agent in the system maintains a list of neighbors and their expertise. Thus, when looking for a piece of information, an agent sends the query to a number of its neighbors who try to answer the query. If they cannot, they send back referrals pointing to other agents that based on their expertise may have the desired information.The algorithm uses the similarity between a query vector and a neighbor expertise vector, as well as its historical referring performance, as the criteria for selecting the next agent in the search process. The use of routing indices-like approaches allows agents to answer queries in an efficient way as information about the interests and abilities of others is collected, without modifying the topology or imposing an overlay structure to the network [43], as long as the number of agents does not increase to very large numbers.

Other approaches use biologically inspired techniques to locate and organize resources [27, 16]. Computer science is interested in understanding how nature is able to self-organize by considering only local information [29, 38]. There are some approaches based on Ant Colony Optimization algorithms (ACOs). 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 entities leave in the environment. This information is taken into account by other agents in order to make decisions. An example of this mechanism for self-organization is presented by Caro *et al.* [12]. They propose a hybrid protocol for packet routing and for improving the efficiency of the paths.

There are other approaches where the underlying topology of the system is loosely structured using certain criteria. This fact facilitates the search process. For instance, Semantics has been included in the systems as a criterion to establish links, to guide the search process or to improve the accuracy of the results. An example of this is presented in [18]. The authors present an overlay structure and routing criterion based on semantic similarity between peers. For selecting the expert peers, semantic topology analysis is exploited. However, the semantic topology is fixed and it is not adapted any further during the lifetime of the experiments. There are other approaches such as INGA [28], where metaphors from on-line social networks as well as semantic similarity between queries and meta data of annotated documents are used in order to improve the search performance. Zhang *et al.* [50] introduce a topology reorganization that group peers with similar data elements. They consider a hierarchical structure that groups agents into levels, and again by groups, based on content similarity. Their search algorithms improve basic flooding approaches by assuming a cooperative system. However,

hierarchical topologies are less flexible in dynamic environments and can create bottlenecks when the number of queries increases.

Reinforcement Learning has been used in the context of information sharing and to dynamically self-organize the structure of the systems. In the proposal presented by Zhang *et al.* [51], the authors describe a reinforcement learning approach for improving the performance of distributed IR search algorithms. Agents maintain an estimated utility by learning from the feedback information returned from previous search sessions. They authors apply the reinforcement learning approach in a hierarchical overlay network based on the content similarity measure among the documents of the agents. In the approaches that make use of reinforcement learning techniques in self-organization, agents adapt their behavior by calculating a probability that is based on information related to their current state, previous decisions, and environment conditions. Abdallah *et al.* [1] present a self-organization mechanism. When an agent receives a message, it updates its current state using a reinforcement algorithm and decides if it is appropriate to stochastically reorganize its current links by 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 slow down learning when moving away from a stable policy and speed up learning when moving towards a stable policy. This approach improves previous proposals based on reinforcement learning [34] 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 on the average degree of connection of the network structure. There are other approaches that combine trust and Q-learning algorithms to self-organize the system structure. Trust has been used to consider agents opinion in order to select candidates to adapt their relations. Q-learning algorithm is used to evaluate the rewards about adapting structural relations [48].

There are other approaches that focus on how systems can be rearranged in order to improve their performance as the environmental conditions and the organizational goals change. For instance, Kota *et al.* [25] present a reorganization mechanism that takes into account different organizational relations between agents that regulate agents interactions. Agents change their relations in order to distribute their workload to subordinates. The change of relations is based on a utility function that considers the reorganization cost, the load of the agent, and the communication cost. The proposed organization model makes some assump-

tions that reduce its application in open environments. In the model, all the agents are acquainted with each other, and an agent has information about its own services, the services provided by its peers, and the accumulated service sets of its subordinates. Therefore, agents rely on a global view of the system for making adaptation decisions.

Kamboj *et al.* [22] present an approach that is based on Organizational Self-Design (OSD) in order to come up with a suitable organizational structure at runtime. The proposed approach is based on two operators (spawning and composition) and a task environment. The reorganization mechanism relies on tasks that are hierarchically organized with regard to the roles of the agents. The decision to reorganize is based on the load of the agent and the last organizational change. If the agent decides to change the organizational structure, it has two possible actions: spawning, which creates new agents to delegate part of its tasks to; or composition with another agent if the agent has been idle for an extended period of time. The main drawback is that this approach relies on a predefined hierarchical structure of tasks that establishes a set of structural adaptation actions.

Gaston *et al.* [15] present a framework for Agent-Organized Networks (AONs) that incorporates adaptation techniques that improve system performance. They propose a bottom-up approach that focuses on the links between agents. The goal of the agents is to maximize their utility, and they are completely selfish. The framework determines when to adapt (if the expected change in utility is over a certain threshold) and which connections to rewire (depending on the number of trades made using this connection). However, the new neighbor that replaces the useless one is selected taking into account broader knowledge that goes beyond its local knowledge about agent's direct neighbors.

In this article, we present a service discovery system with self-organization mechanisms that attempts to improve previous approaches in several ways. First, it completely relies on the local information of agents about the attributes of their direct neighbors and the information generated as the result of interactions during the service discovery process. Agents do not rely on a hierarchical organization in order to make decisions about the search process or the self-organization. Therefore, the system provides robustness, scalability, and facilitates the self-organization. Second, our model focuses not only on the analysis of structural relations between agents but also on the analysis of the population of the system. Third, agents are able to reason about when it is most appropriate to consider a local self-organization action. Agents determine: (i) whether or not their position in the system is beneficial; (ii) whether or not it is worthwhile to change their structural relations with neighbors; (iii) and which acquaintances are the most suitable

for establishing a profitable relation with (i.e., the set of potential neighbors is not random, it is established taking the activity of the system into account). Fourth, the system integrates semantic information as well as information from previous searches in the self-organization process. Finally, experiments that evaluate our approach and compare it with other existing proposals are presented.

3. Service Discovery Scenario

Currently, there are more and more web services available that allow customers to deal with their daily activities: communications (mail, news, social), search (multimedia, item), entertainment (travel, sports, weather), or work tasks on-line [7]. As the number of services increases, there is a need to facilitate location and coordination among them [32]. Moreover, the available services change dynamically, and customers usually do not have a global view of the functionality of the whole system. Thus, mechanisms that facilitate the location of these services in a decentralized way are required. Furthermore, web activity evolves over time, for example according to the time of the day, the different days of the week, or different seasons of the year [19, 3]. For this reason, the system should be able to adapt itself without external coordination according to customer demand in each moment.

To illustrate the context where the self-organization mechanisms are applied, let us present a service discovery scenario where the discovery process is described as well as the situations where self-organization mechanisms are applied. Consider a network of services as a form of autonomic cloud computing system. This network contains different groups of semantic web services provided by software agents as part of an overlaying network. In some situations, these agents should interact with each other to achieve a task that they cannot afford 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 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.

The scenario in Figure 1 shows a network of a few agents that play organizational roles and offer semantic web services. The structural relations between these agents have been established taking a probabilistic criterion based on choice homophily into account [11]. Choice homophily determines the similarity of two agents considering their attributes (i.e., the role an agent plays and the services

and agent offers). Therefore, agents have more connections with agents that play similar roles and offer similar services than with dissimilar ones. In the case of agent i, it has connections with agents k and j, which play similar roles and offer similar services, and agent n, which plays a dissimilar role and offers a dissimilar service. Note that agents that play similar roles are represented in Figure 1 with similar colors. Agent i offers the service s_1; however, in order to achieve one of its goals, it needs to locate an agent that offers a service similar to s_6 and plays a role similar to r_5. At that moment, agent i creates a query $q = \{s_6, r_5\}$ that consists of the required semantic service description and the organizational role that the target agent should play. The query has a Time To Live (TTL) associated, which is the maximum number of times that it can be forwarded. 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 shown in Figure 1a , agent i should choose one of its neighbors, n,j, or k, to forward the query q. In order to select the most promising neighbor, the agent i applies a function that considers: (i) the choice homophily between the neighbors of agent i and a *fictitious* agent 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 1 ($CH(k,t) = CH(j,t) = 0.5$, and $CH(n,t) = 0.15$) and their degrees of connection, agent i sends the query to the most promising agent (i.e., agent 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 v (see Figure 1a). Afterwards, agent i stores agent v in its local view as a possible candidate for establishing a future structural relation if some of its current relations are not useful.

We assume that, as time passes, service demand changes and the services offered by agents that play the role r_5 start to be the most demanded services in the system. As a result, agent i analyzes its internal state st_i and realizes that its structural relation with neighbor n has become useless. Therefore, agent i decides to break its current structural relation with n and establish a new one with a candidate that was discovered as a result of a previous search process ($(i,n) \rightarrow (i,v)$) (see Figure 1b). This self-organizative action reduces the distance towards the agents that provide the most demanded services and improves the success rate in future discovery processes.

Also, there are other agents (such as agent m) that, through an analysis of

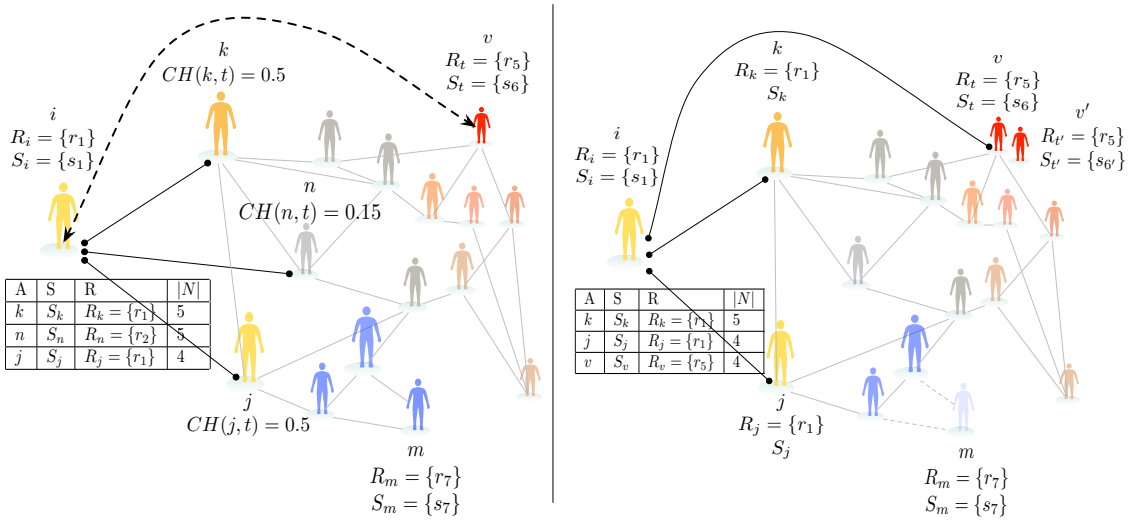


Figure 1: An example of a decentralized service discovery system.

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 1, agent 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. In our scenario, assuming that the demand for services offered by agents with role r_5 increases considerably, agent v creates a clone, 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.

4. Formal Model

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 useless

structural relations with profitable ones or to leave, continue, or clone themselves in order to adapt the system population to the service demand. In this section, we introduce the notation of the main components that are part of the decentralized service discovery system.

The model presented in this work is based on a set of autonomous agents that play at least one organizational role and offer their functionality through a set of semantic services. These agents have a reduced view of the global community: just a limited number of direct neighbors are known and the rest of the network remains invisible to them.

DEFINITION **1.** *(System). The system is defined as a Service-Oriented Multi-Agent System* $SOMAS = (A, L)$*, where* $A = \{i, ..., n\}$ *is a finite set of autonomous agents that are part of the system, and* $L \subseteq A \times A$ *is the set of links, where each link* $(i, j) \in L$ *indicates the existence of a direct relationship between agent* i *and agent* j *based on the homophily criterion.*

It is assumed that the knowledge relationship between agents is symmetric, so the network is an undirected graph.

An agent is a social entity that interacts with other agents in the system. It controls its own information about (i) the semantic services it offers, (ii) the organizational roles it plays, and (iii) local knowledge about its immediate neighbors. The agent is unaware of the rest of the agents in the system.

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

- $R_i = \{r_1, \dots r_m\}$ *is the set of roles played by the agent;*
- $S_i = \{s_i, \dots, s_n\}$ *is the set of services provided by the agent. Each service should be associated to at least one of the roles played by the agent,* $s_i \in \bigcup_{\forall r_i \in R_i} S_{r_i}$.
- N_i *is the set of neighbors of the agent,* $N_i = \{m, ..., p\} : \forall j \in N_i, \exists (i, j) \in L$*, and* $|N_i| > 0$*. It is assumed that* $|N_i| \ll |A|$*;*
- st_i *is the internal state of the agent;*

Next, we are going to describe with more detail two of the attributes that are part of an agent in our system: the role and the internal state.

Agents located in our system play at least one organizational role. The organizational role of an agent determines the type of services that it can offer.

DEFINITION **3.** *(Role). A role $r_i \in R_i$ is defined by the tuple (ϕ_i, S_i) , where:*

- *ϕ_i is a semantic concept for the role;*
- *$S_{r_i} = \{s_1, \ldots, s_l\}$ is the set of services associated to the role. Each service is defined by the tuple $s_i = (I_i, O_i, P_i, E_i)$, where the components are the set of inputs (I_i), outputs (O_i), preconditions (P_i), and effects (E_i) of the services, respectively. All of them are semantic concepts that can be defined in different ontologies.*

Each agent in the system maintains a local view of what is happening. This local view is stored in its *internal state*, st_i, 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 [4].

DEFINITION **4.** *(Internal State). The internal state of an agent in our system st_i is characterized by a tuple of $(\mathcal{K}_i^N \cup \mathcal{K}_i^A, \mathcal{K}_i^E, \mathcal{K}_i^{st})$ where:*

- *$\mathcal{K}_i^N \cup \mathcal{K}_i^A$ is the partial knowledge about the neighborhood of agent i;*
- *$\mathcal{K}_i^E$ is the model of the local environment;*
- *$\mathcal{K}_i^{st}$ is the status of the agent.*

The $(\mathcal{K}_i^N \cup \mathcal{K}_i^A)$ 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 $\mathcal{K}_i^N =< \{R_j, |N_j|, Q_{ij}\} \forall j \in N_i >$. Agent i contains information about each neighbor j: the roles j plays, the services j offers, the degree of connection of j, and the number of times that a query that arrived to the agent i was not forwarded through its neighbor j;

- a set of acquaintances $\mathcal{K}_i^A$ whose existence agent i is at least aware 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.

The $\mathcal{K}_i^E$ 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 position in the system. In the case of our service discovery scenario, an agent maintains the following information $\mathcal{K}_i^E =< \overline{q_i}, Q_i, \rho_i, >$ (see Figure 2a), where:

- $\overline{q_i} = [q_i^{r_1}, q_i^{r_2}, ...,]$ is the local service demand distribution (i.e., the number of queries that the agent receives about services that are being demanded in the system);

- Q_i is the number of total queries that the agent receives;

- ρ_i is the correlation coefficient that establishes the relationship between the local service demand distribution $\overline{q_i}$ and an estimation of the expected service demand distribution.Power-law, Exponential, and Zipf's-law distributions are present in many features of Internet [2, 20]. 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 $\overline{q_i}$. 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.

The $\mathcal{K}_i^{st}$ 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 state. 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 state. It is important to determine the state 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.

Besides the attributes that an agent in our system has, it also has a set of functions to deal with the dynamics of the system: $\mathcal{F}_N$ and $\mathcal{F}_A$. The first function $\mathcal{F}_N : A \rightarrow N_i$ is used in the decentralized discovery process. This function calculates the most promising neighbor j of an agent i to forward a query $q = \{s_q, r_q\}$. The query contains the description of the service, s_q, and role desired , r_q, towards an initially unknown target agent that solves it:

$$\mathcal{F}_N(t) = \underset{j \in N_i}{\operatorname{argmax}} \left[1 - \left(1 - \left(\frac{CH(j,t)}{\sum_{n \in N_i} CH(n,t)} \right) \right)^{|N_j|} \right] \tag{1}$$

The parameter t is a *fictitious* agent $t = (r_q, s_q, \emptyset, \emptyset)$ that represents the unknown provider agent that is able to provide a service similar to the service that appears in the query and also a role similar to the role that appears in the query. This formula is based on the choice homophily (CH) [11] between neighbor $j \in N_i$ and agent t, which estimates how close is agent j to the unknown provider agent t. The divisor of the expression is just a normalization factor. The degree of connection of a neighbor j is also considered in the formula ($|N_j|$). The consideration of the degree is also important in the discovery process since highly connected agents have more neighbors and, therefore, they have more probabilities to interconnect different communities. If there are two neighbors that have the same degree of similarity, the function selects the neighbor with highest degree of connection.

The second function $\mathcal{F}_A : st_i \rightarrow \Psi$ is used to decide local self-organization actions based on the internal state of the agent, where $\Psi = \{$*clone, remain, leave,*

rewire$\}$ is the set of possible actions.

Once we have defined the components of the system, we describe briefly how the relations between agents are established. Note that our focus is not on describing how the system structure is created. Rather, it is on self-organization mechanisms to improve the service discovery performance. However, in order to facilitate the understanding of how the structure of the system is created and how the service discovery process works, we introduce a brief description of how relations between agents are established. For a detailed mathematical treatment about this process, we refer the reader to [11].

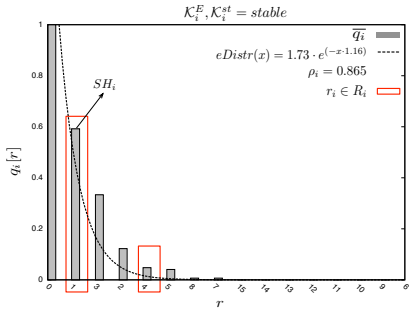
Agents establish structural relations with other agents based on choice homophily (CH) as a self-organization criterion [26]. The choice homophily concept, translated to the agent context, has been considered as the similarity measure based on the service description and organizational role. Agents have more probability to establish links with similar agents (i.e., the degree of choice homophily between two agents is over a threshold ($CH(i,j) > \varepsilon$)) than with dissimilar ones. When a new agent, i, arrives to the system, it establishes at least one structural relation with another agent, j, that is already present in the system. Each agent that is part of the system is considered an entry point. If an external agent i wants to get into the system, it follows the following steps:

- Initially, agent i should know at least one agent j in the system. Agent i sends a request to agent j to be part of the system.

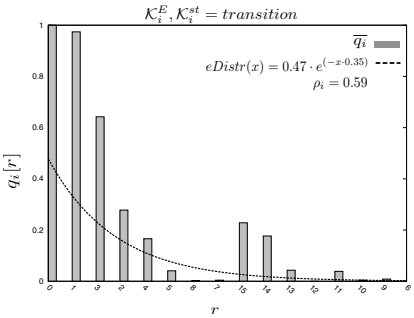
- If j sends i a *refuse message* the interaction finishes. Otherwise, j allows i to get into the system and it sends i an agree message. This means that j, based on the choice homophily between them, is going to consider the establishment of a link with i.

- If agent j decides to establish a link with i, it sends an *inform message* to i with the link information ($\langle i,j \rangle$). Otherwise, j forwards the request to one of its neighbors k randomly selected. The process is repeated until agent i receives an *inform message* with its neighbor ($\langle i,k \rangle$) 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

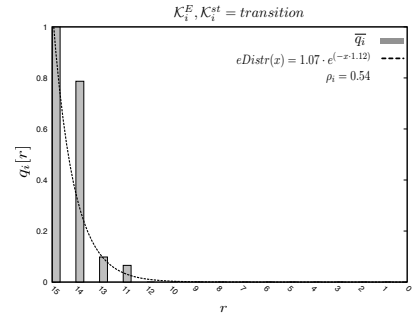
between dissimilar agents allow agents to locate other agents communities in a few steps.



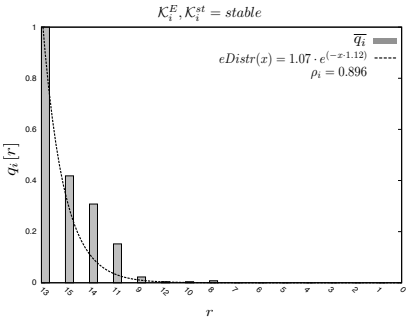
(a) Initial st_i adapted to the current service demand. The agent fixes the distribution of roles. ρ_i is near 1 and $\mathcal{K}_i^{st}$ is *stable*.



(b) st_i when service demand changes. Services offered by agents that play roles r15, r14, r13, and r11 start to be demanded. ρ_i decreases 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 close to 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 2: Example of the internal state st_i of the agent i. Each plot shows the local view of the service demand distribution $\overline{q_i}$ in a certain moment. The x-axis shows the numeric identifiers for the roles $(r_0, r_1, ...)$ that appear in the queries that agent i receives. The y-axis shows the number of queries forwarded of each role $\overline{q_i}[r]$. 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.

5. Self-Organization for Service Discovery

Agents reason about local properties of the system by combining their *partial view of the neighborhood*, their *model of the local environment*, and their *internal*

status. As a result of this reasoning process, agents make local decisions that are translated into actions, $\mathcal{F}_{Ai} : st_i \rightarrow \Psi$. 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.

Algorithm 1 shows an overview of the reasoning process followed by the agents during the discovery process, where self-organization mechanisms are considered. This process is initiated when agent i generates a query $q = (s_q, r_q)$, which contains the semantic description of the desired service and the role that the target agent should play. A *fictitious* agent t is created with the service and role specified in the query q (Line 3). Agent i looks for a neighbor similar to t. If it finds a suitable neighbor, the service discovery process ends (Lines 5-10). Otherwise, the agent i forwards q to one of its neighbors $j \in N_i$. Specifically, q is forwarded to the agent that has semantic closeness (degree of choice homophily) to the *fictitious* agent t and also has a high degree of connection (see Equation 1)(Line 12). Then, agent i updates its information about which of its links have been used (Line 13). Agent i also updates the number of total queries it received (Q_i), and the number of queries about the role r_q ($\overline{q_i}[r_q]$). If the query is solved, the source agent (which started the service discovery process) adds the provider agent that was found to its set of acquaintances if the source agent does not already have another agent in its acquaintances that plays the same role (Lines 19-20). Finally, the source agent updates its internal state st_i and analyzes the set of self-organization actions that it can carry out.

In the following sections, we are going to describe the main functions related to self-organization that appear in the service discovery algorithm: *Internal state analysis*, *Structural adaptation*, and *Agent adaptation*.

5.1. 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 know what is happening in the system (see Algorithm 2).

Initially, if the agent is in a *transition* state, it can reorganize the local service demand distribution $\overline{q_i}$ 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 2c and 2d where agent i is in a *transition* state and decides to reorganize the local service demand distribution $\overline{q_i}$ ($\{q^{r_{15}} \gg q^{r_{14}} \gg q^{r_{13}} \gg q^{r_{11}} \gg q^{r_{12}} \gg$

Algorithm 1 Function that describes the service discovery process where self-organization mechanisms are considered.

```
1: function SelfAdaptationServiceDiscovery(i, q = (s_q, r_q))
2: source ← i
3: t ← (r_q, s_q, ∅, ∅)
4: while ¬found ∧ TTL ≤ 100 do
5:     for j ∈ K_i^N do
6:         if CH(j, t) > ε then
7:             found ← true
8:             target ← j
9:         end if
10:    end for
11:    if ¬found then
12:        n ← F_N(t)
13:        updateLinksDecay(i, n)
14:        q̄_i[r_q] ← q̄_i[r_q] + 1
15:        Q_i ← Q_i + 1
16:        TTL ← TTL + 1
17:        i ← n
18:    end if
19:    if found ∧ ∄k ∈ K^A_source : r_q ∈ R_k then
20:        K^A_source ← K^A_source ∪ target
21:    end if
22: end while
23: InternalStateAnalysis(source)
24: StructuralAdaptation(source)
25: AgentAdaptation(source)
26: end function
```

$q^{r_{10}} \gg q^{r_9} \gg q^{r_8}\}$ 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 $\overline{q_i}$ 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 change its status $\mathcal{K}_i^{st}$ to *transition*, the agent should receive a sufficient number of queries. The significance of the number of received queries is evaluated through a logistic function (Lines 10-24):

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

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).

Any of these states 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.

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.

5.2. Structural Self-Organization

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, they consider a *decay* metric that it is associated to each link. This metric indicates the probability of maintaining the 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:

Algorithm 2 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: $a, b \leftarrow leastSquaresFitting(\overline{q_i})$
6: $\rho_i \leftarrow linearCorrelation(a \cdot e^{(b \cdot x)}, \overline{q_i})$
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**

$$decay(Q_{ij}) = 1 - \frac{1}{1 + \cdot e^{\frac{-(Q_{ij} - z)}{y}}}, \tag{3}$$

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 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 make changes in its current links. Q_{ij} is the number of queries that arrived to agent i and were not forwarded through agent j. Every time that agent i forwards a query, it updates the information about the usage of its links. If the query is forwarded through agent j, the Q_{ij} is updated to 0. Otherwise, the Q_{ij} is increased by increments of 1 (see Algorithm 1, Line 13). With the information provided by the *decay* function, agent i reasons about the benefit of maintaining its current links.

In Algorithm 3, we describe the reasoning process that agents follow to adapt their current links. Each agent invokes this function when ends a service discovery process that the agent has initiated. If the agent has at least one acquaintance

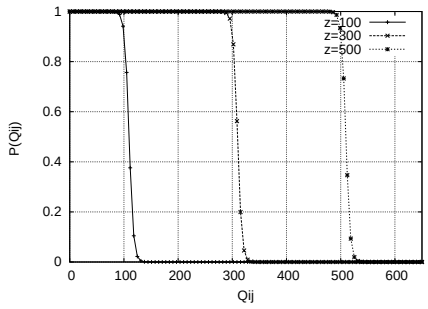


Figure 3: Decay function for the structural links of agents with different values for the displacement parameter z.

($|\mathcal{K}_i^A| > 0$) 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 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 $\mathcal{K}_i^A$ that was known as a result of a previous service discovery process. 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).

5.3. Population Self-Organization: Leaving, Remaining, or Cloning

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

- the number of queries received by the agent Q_i (see Equation 2),
- the degree of correlation ρ_i,
- the structural homophily of the agent SH_i.

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

Algorithm 3 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(i)
2:  if |K_i^A| > 0 then
3:      for j ∈ K_i^N do
4:          decay ← decay(Q_ij)
5:          if decay < random then
6:              while ¬found ∧ 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 − n
13:                 K_i^N ← K_i^N ∪ n
14:                 K_i^N ← K_i^N − j
15:             end if
16:         end if
17:     end for
18: end if
19: end function
```

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 $\overline{q_i}$. Then, it 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). The structural homophily of an agent with respect the system dynamics is defined by the following function:

$$SH_i(x) = a \cdot e^{x \cdot b} \tag{4}$$

where x is the role of agent i that maximizes the following function:

$$\underset{x \in R_i}{\operatorname{argmax}}\, a \cdot e^{x \cdot b} \tag{5}$$

where the a and b parameters are obtained in the *InternalStateAnalysis* function (see Algorithm 2 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 i is calculated is shown in Figure 2a. Agent i plays two roles: r_1 and r_4. At that moment, using the data

in $\overline{q_i}$, 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 i is:

$$SH_i(x) = 1.73 \cdot e^{x \cdot 1.16}, \tag{6}$$

where x is the role that maximizes the function SH_i

$$\underset{r_1, r_4 \in R_i}{\operatorname{argmax}}[1.73 \cdot e^{-r_1 \cdot 1.16}, 1.73 \cdot e^{-r_4 \cdot 1.16}] = \underset{r_1, r_4}{\operatorname{argmax}}[0.54, 0.018] \tag{7}$$

Therefore, the value of x that maximizes the function for the structural homophily SH is r_1. Then, x is substituted by r_1 in equation 6 and we obtain the *structural homophily* of agent i ($SH_i = 0.54$). This value means that the services that agent i offers are being demanded, but are not the most demanded services in the system.

The population adaptation algorithm (see Algorithm 4) evaluates whether or not it is worthwhile for an agent to remain in the system. 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 *structural homophily* 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. Each neighbor will start a search process of a similar agent to the previous one to establish a new link with it.

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:

$$P(Q_i, clones) = \frac{1}{1 + \cdot e^{\frac{-(Q_i - 2^{clones})}{y}}}, \tag{8}$$

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 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 probability of cloning decreases exponentially over time. 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 conditions, 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. The new agent will follow the steps described at the end of Section for new agents that arrive at the system and want to be part of it. When an agent creates a clone it resets its internal state.

Algorithm 4 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.

```
1:  function PopulationAdaptation(i)
2:  pLeave ← 1/(1 + ·e^{−(Q_i − d')/y})
3:  if ((K_i^{st} = STABLE) ∧ (SH_i < random) ∧ (pLeave > random)) then
4:      if (similarN(N_i) >0) then
5:          leave()
6:          leave ← true
7:      end if
8:  end if
9:  pClone ← 1/(1 + ·e^{−(Q_i − 2^{clones})/y})
10: if (K_i^{st} = STABLE) ∧ (ρ_i > δ) ∧ ¬leave
    ∧(SH_i) > random)) then
11:     if ((similarN(N_i) <| N_i |) ∧ (Δq^i > 0)) then
12:         clone()
13:     end if
14: end if
15: end function
```

6. Results

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 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 [21] [35]. 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.

$$CL = 1 - \frac{\text{number of messages generated in a self-organized system}}{\text{number of messages generated in a system that is not self-organized}} \quad (9)$$

- 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 specified Time To Live ($TTL = 100$). The TTL parameter determines the maximum number of forwarding actions allowed in the system for a concrete query. We evaluated how the variation of the value of this parameter affects to the discovery process using different search strategies (see Figure 4). As it was expected, higher values of TTL offered the opportunity to improve the success rate. In the experiments, we considered a $TTL = 100$ to give the chance to reach the target. 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 [2, 20]. If agents play a role and offer a service that are similar to the most demanded roles and services in the system, then they have a low probability of generating queries that are dissimilar to them. Otherwise, agents that play a role and offer a service that are not similar to the most demanded roles and services have a high probability of generating dissimilar queries to their service provision abilities. Once the system population is self-adapted to the service demand (i.e., agents that provide services that are not demanded leave the system and agents with demanded services create clones), the majority of the agents have a low probability of generating dissimilar queries to their service abilities. In the experiments, we made a snapshot of all the metrics every 10,000 queries in order to see the evolution of the system.

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

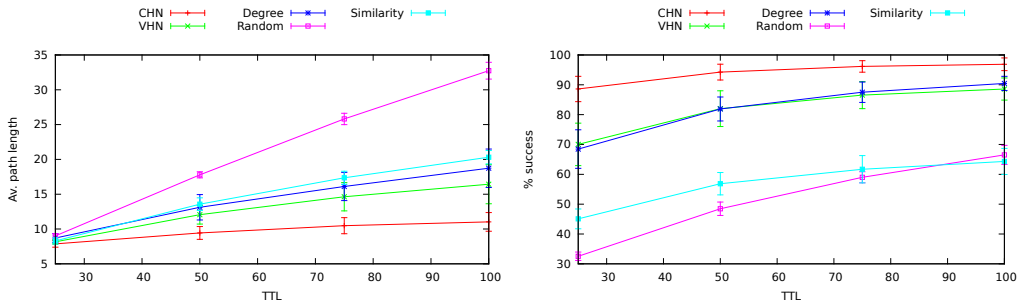


Figure 4: Influence on the average path length and on the success of the TTL parameter when different search strategies are used. The difference among them is how the most promising neighbor is selected in each step (Random: a search process that uses random walks; Degree: a search process that uses only degree of connection information; Similarity: a search process that uses only service similarity information; VHN: a mixed search process that uses a combination of degree of connection and service similarity; CHN: a mixed search process that is based on the degree of connection, and service and role similarity). 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.

6.0.1. 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)[1]. We chose this algorithm to compare our approach because RL is a common approach for solving multi-agent decision problems and specifically, the proposal presented in [1] 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 [6]. As in many RL algorithms, WPL makes use of two matrices, π_i and Q_i, for each agent. In the the context of service discovery, π_i represents the neighbors of an agent and their suitability to forward queries of certain type. Thus, the organizational roles are the states of π_i and the actions are the neighbors. The states are updated with the feedback obtained from the routing process. 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 3). 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.

6.1. First Scenario: static service demand.

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 an average degree of connection $d = 4$ because with `RL-based` mechanism a lower value divided the network into isolated parts.

Each graph has associated a table that contains the results obtained in certain snapshots sn, including the error interval. 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 in the graphs.

Figure 5 shows that the introduction of self-organization mechanisms considerably reduces the number of steps required to reach a suitable provider agent. The x-axis represents the snapshots, and the y-axis indicates the average number of steps. 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 (see Table 1 (Left)). In these configurations, agents are prone to change their structural

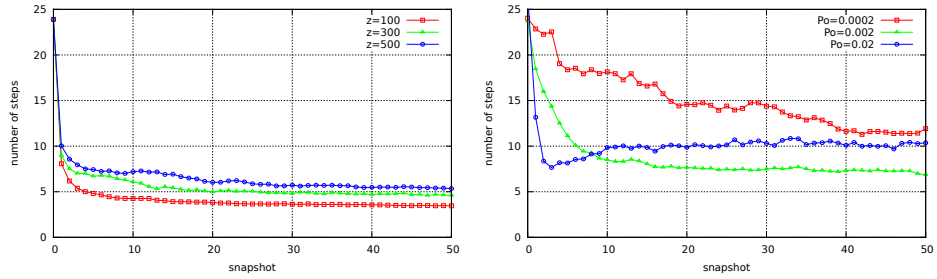


Figure 5: Average path length obtained when agents use link-based adaptation mechanisms: (Left) `Decay-based`, (Right) `RL-based`.

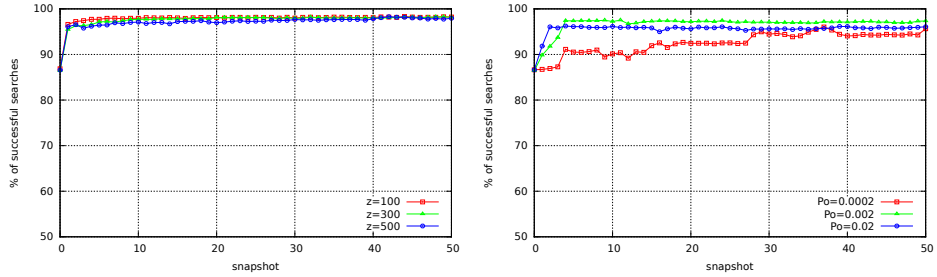


Figure 6: Percentage of successful searches (searches that are solved before the TTL) when agents use link-based adaptation mechanisms: (Left) `Decay-based`, (Right) `RL-based`.

	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	$\mathbf{6.87 \pm 0.53}$
$P_o = 0.02$	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 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.

links. The `Decay-based` method is less sensitive to configuration parameters than `RL-based`, and, in general, the improvement in the average path length for the `Decay-based` method was better than the improvement obtained with `RL-based` mechanism.

Figure 6 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 7 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 of the variations in the service demand; therefore, they realized that structural changes were needed to adapt some of their links according to the new situation. 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 8 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 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 modify 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 configura-

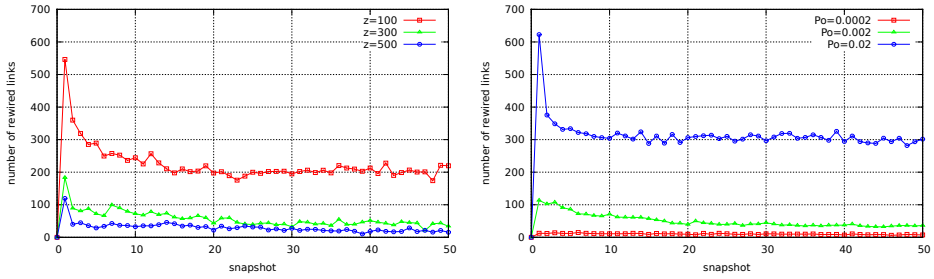


Figure 7: Number of structural relations rewired when agents use link-based adaptation mechanisms: (Left) `Decay-based`, (Right) `RL-based`.

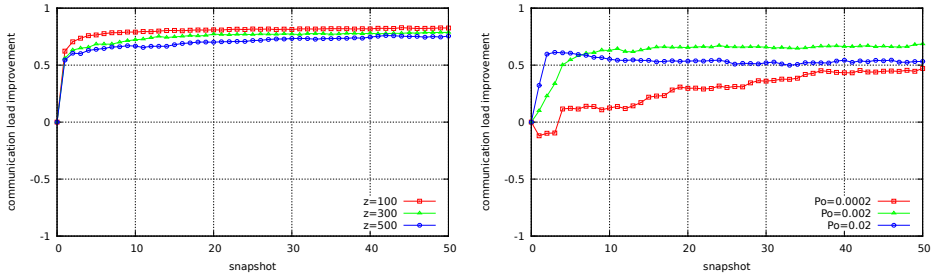


Figure 8: Communication load metric when agents use link-based adaptation mechanisms: (Left) `Decay-based`, (Right) `RL-based`.

tion cause 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.

6.2. 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

	Number of rewired links		
	$sn = 1$	$sn = 5$	$sn = 50$
$z = 100$	546.2 ± 37.44	289.4 ± 62.38	219.8 ± 78.25
$\mathbf{z = 300}$	$\mathbf{183.4 \pm 41.73}$	$\mathbf{72.2 \pm 20.68}$	$\mathbf{34.0 \pm 14.45}$
$z = 500$	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
$\mathbf{P_o = 0.002}$	$\mathbf{113.4 \pm 28.17}$	$\mathbf{86.0 \pm 4.11}$	$\mathbf{36.2 \pm 8.16}$
$P_o = 0.02$	622.6 ± 91.84	333.6 ± 35.59	301.4 ± 47.40

	Communication load improvement		
	$sn = 1$	$sn = 5$	$sn = 50$
$\mathbf{z = 100}$	0.62 ± 0.02	0.76 ± 0.03	$\mathbf{0.82 \pm 0.01}$
$z = 300$	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
$\mathbf{P_o = 0.002}$	0.09 ± 0.45	0.54 ± 0.20	$\mathbf{0.68 \pm 0.039}$
$P_o = 0.02$	0.32 ± 0.39	0.60 ± 0.17	0.53 ± 0.10

Table 2: (Left) Number of structural relations rewired, and (Right) Communication load improvement in different snapshots sn when agents use link-based adaptation mechanisms: `Decay-based`, and `RL-based`.

system evolution. There were two intervals. In previous works, we tested the self-organization of the system with different query distributions [10]. In these experiments, we considered the exponential distribution since it is commonly present in large-scale service environments [2, 20]. 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. We considered this new distribution for the second interval in order to see the behavior of the mechanisms in the worst scenario (i.e., the scenario where the distribution is the opposite to the previous one). In this second scenario, we only show the results obtained with the best configurations of each mechanism: for `Decay-based` mechanism $z = 300$ and $z = 100$, and for `RL-based` mechanism $P_o = 0.002$.

Figure 9 (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 sys-

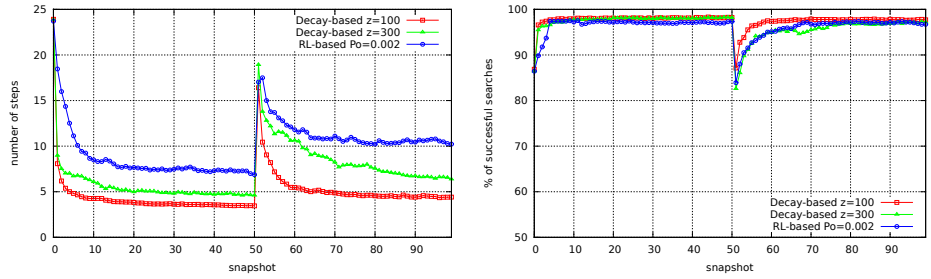


Figure 9: (Left) Average number of steps required to reach the desirable provider agent, and (Right) percentage of successful queries when there are dynamic changes in the service demand and agents use link-based adaptation mechanisms: `Decay-based` and `RL-based`.

tem 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 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 3).

Figure 9 (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 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 10 (Left) shows the number of rewired structural relations to deal with variations in the service demand distribution at each moment. For `Decay-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 3: (Up) Average path length, and (Down) Percentage of successful searches in different snapshots sn when service demand changes and agents use link-based adaptation mechanisms: `Decay-based`, and `RL-based`.

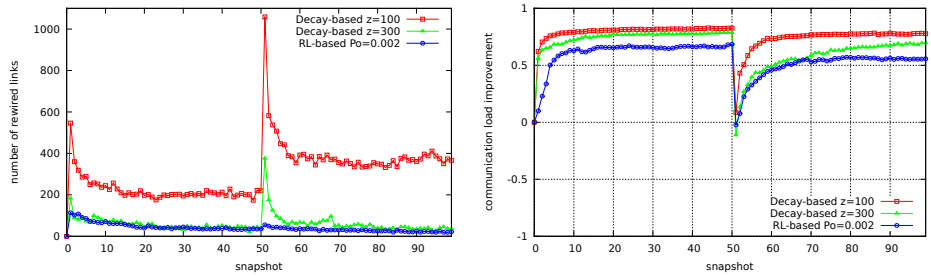


Figure 10: (Left) Number of structural relations rewired, and (Right) Communication load improvement when the service demand changes and agents use link-based adaptation mechanisms: `Decay-based` and `RL-based`.

	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
$\mathbf{z = 300}$	$\mathbf{183.4 \pm 41.73}$	72.2 ± 20.68	$\mathbf{34.0 \pm 14.45}$	$\mathbf{376.0 \pm 91.58}$	85.0 ± 53.86	$\mathbf{35.8 \pm 31.39}$
$P_o = 0.002$	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
$\mathbf{z = 100}$	0.62 ± 0.02	0.76 ± 0.03	$\mathbf{0.82 \pm 0.01}$	0.086 ± 0.12	0.64 ± 0.05	$\mathbf{0.77 \pm 0.02}$
$z = 300$	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 4: (Up) Number of structural relations changed, and (Down) Communication improvement in different snapshots sn when service demand changes and agents use link-based adaptation mechanisms: `Decay-based`, and `RL-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 aware that there was a change in the services that were being demanded in the system, and, therefore, there were links that started to be useless. Agents with parameter $z = 100$ were more prone to changes, and they did not wait very long to decide to rewire these useless relations 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 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 10 (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

	Network properties			
	prop.	sn=1	sn=50	sn=99
$z = 100$	*CH*	0.57	0.18	0.17
	C	0.032	0.011	0.008
	k	4.25	4.30	4.47
	D	14.0	11.39	10.19
	p	5.95	4.61	4.67
$z = 300$	*CH*	0.57	0.20	0.19
	C	0.032	0.008	0.0049
	k	4.25	4.27	4.33
	D	14.0	11.0	10.80
	p	5.95	4.77	4.71
$P_o = 0.002$	*CH*	0.57	0.45	0.53
	C	0.0328	0.017	0.017
	k	4.25	4.70	5.39
	D	14.0	10.39	8.60
	p	5.95	4.90	4.59

Table 5: Network properties in different snapshots sn when agents use different adaptation mechanisms, `Decay-based`, and `RL-based`, and the service demand changes. The network properties are: *CH* choice homophily, C clustering coefficient, k average degree of connection, D diameter, and p average path length.

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`.

Network properties change as the self-organization process evolves. In Table 5, we show the properties in certain snapshots. It can be observed that as the agents adapt their links considering the service demand, the choice homophily in the network decreases as well as the clustering coefficient. This fact is due to the fact that agents establish new links with other agents that play demanded roles, and they do not consider homophily in that process. However, there is still a degree of homophily in the network. Another consequence of the self-organization in the network structure is that the average degree of connection and the diameter are reduced. The effects of self-organization in the network structure are less important in networks where agents use `RL-based` mechanisms since less structural changes are made by the agents. Regarding the structural properties of the network, we can conclude that the network structure looses a certain degree of homophily in order to adapt itself to the service demand. The effects of the self-organization in the network structure reduce its diameter and the average path length.

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.

6.3. 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 were defined, changing the distribution of the demanded services. 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 11 (Left) shows the effects of local decisions of agents about their position in the network on 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 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

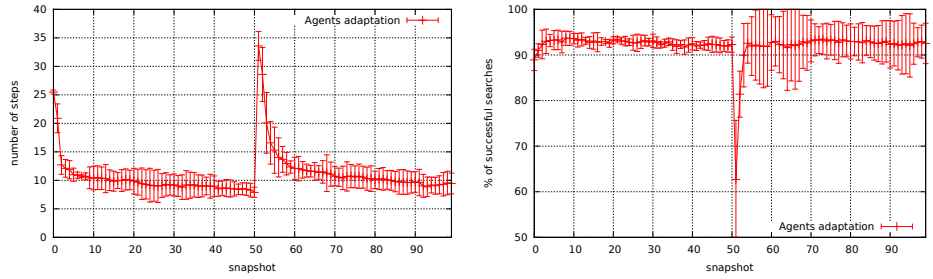


Figure 11: (Left)Average number of steps required to locate the provider agent of the required service, and (Right) Percentage of successful queries when agents use `Population-based` adaptation mechanism.

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 position in the system reduced the number of steps required to reach the provider agent needed.

Figure 11 (Right) shows the effects of local decisions of agents about leaving or cloning themselves in the service discovery success. The `Population-based` mechanism considerably increased the percentage of successful searches, and it was able to deal with service demand changes. Note that, in the second interval, the 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, a higher variability in the system adaptation was introduced and, consequently, in the success of the service discovery process.

Figure 12 (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

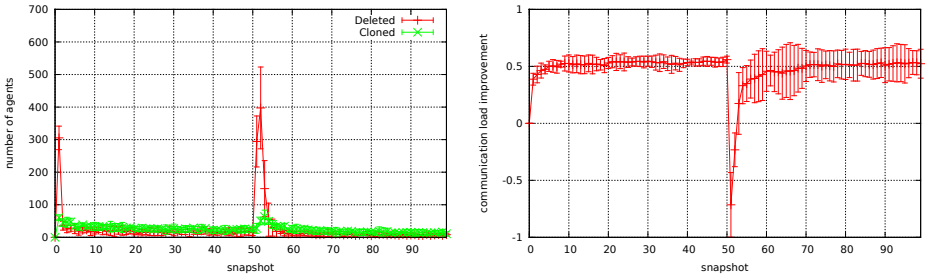


Figure 12: (Left) Number of agents that decide to clone or leave the system, and (Right) Communication load improvement as the service demand distribution changes.

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) gradually diminished as the system became adapted.

Figure 12 (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. 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.

Table 6 shows the properties of the networks in different snapshots when agents use `Population-based` mechanisms for self-organization. The initial network homophily is lower than in previous scenarios since the average degree of connection is lower. The homophily in the network decreases as the system self-organizes its structure according to the service demand. The clustering coefficient does not decrease as much as the homophily degree since there are not so many link changes based on service demand as in previous scenarios. When agents

	Network properties			
	prop.	sn=1	sn=50	sn=99
`Population-based`	*CH*	0.35	0.32	0.22
	C	0.0037	0.0036	0.003
	k	2.53	3.05	3.14
	D	18.60	14.19	12.0
	p	8.25	6.52	5.92

Table 6: Network properties in different snapshots sn when agents use `Population-based` mechanisms and the service demand changes. The network properties are: *CH* choice homophily, C clustering coefficient, k average degree of connection, D diameter, and p average path length.

create a clone or leave the system, agents create and establish new links based on homophily criterion. It can be observed that the self-organization reduces the network diameter and the average path length.

6.4. Combining Self-Adaptation 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 13 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 Table 7). 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

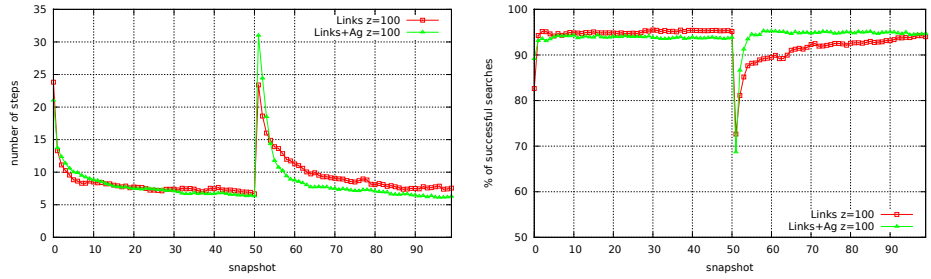


Figure 13: (Left) Average path, and (Right) Percentage of successful queries when agents include `Decay-based` and `Population-based` mechanisms in the discovery process.

	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 7: (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.

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 13 (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 beginning 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 variabil-

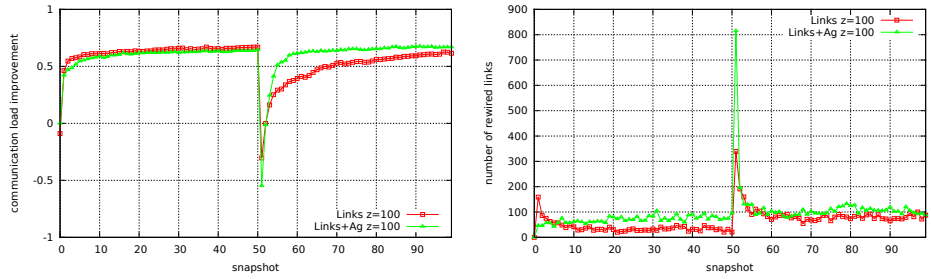


Figure 14: (Left) Improvement in the communication load, and (Right) Number of structural changes when `Decay-based` and `Population-based` mechanisms are combined.

ity in the obtained results, mainly when the changes in the service demand were significant (see Table 7).

Figure 14 (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 8).

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 12 (Left)).

Figure 14 (Right) shows the number of changes in the structural relations between agents. The number of structural changes increased when both mechanisms were 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.

Table 9 shows network properties in different snapshots when agents use `Decay-based` and `Population-based` mechanisms for self-organization. As in previous scenarios, the network homophily decreases as the system adapts itself to the service demand. The clustering coefficient does not decrease since the combination of `Decay-based` and `Population-based` mechanisms generates less changes in the agents links than only `Decay-based`.

Taking into account the results of the different tests, we can conclude that the inclusion of self-organization mechanisms considerably improves the system per-

	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 8: (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.

	Network properties			
	prop.	sn=1	sn=50	sn=99
$Links + Ag$	CH	0.35	0.2	0.2
	C	0.003	0.002	0.003
	k	2.52	3.19	3.2
	D	18.0	14.0	12
	p	8.11	6.37	5.62

Table 9: Network properties in different snapshots sn when agents use `Decay-based` and `Population-based` mechanisms and the service demand changes. The network properties are: CH choice homophily, C clustering coefficient, k average degree of connection, D diameter, and p average path length.

formance. Specifically, in the case of service discovery, self-organization mechanisms introduce an improvement in the percentage of successful searches as well as in the average number of steps required to locate a suitable provider agent. The communication load is also considerably reduced.

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 higher cost for adaptation, reflected in the number of modified structural relations. The number of changes in the structural relations 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, it needs 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.

7. Conclusions

Environmental conditions in open, service-oriented systems can change and the systems should be able to adapt to new circumstances in a decentralized way. The majority of self-organization proposals only consider changes in the structural relations of the systems and do not consider population adaptation or the

combination of both.

In the presented system, we have proposed a model for self-organization. In this model, each agent is able to reason about when it is appropriate to apply self-organization actions based on its local view of the environment. Agents consider not only the adaptation of the structural links, but also the adaptation of the population. The self-organization of the links is based on an estimation of their utility, by analyzing how many times an agent uses each link to forward queries. The links that are not being used are replaced with new structural relations with acquaintances. The acquaintances are not randomly chosen, as it occurs in the majority of proposals present in the literature. The acquaintances are established as a result of the service discovery activity, considering which are the roles that are being demanded in that moment. Regarding to the self-organization of the population, the proposed mechanism analyses whether the services provided by one agent are demanded in the system or not.

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. The combination of `Decay-based` and `Population-based` mechanisms improved the system performance by reducing the latency to achieve a complete adaptation of the system, offering a more reactive behavior and a lower variability in the results in highly dynamic environments.

As future work, we plan to consider costs associated to self-organization actions such as rewiring links or cloning in order to analyze how this fact influences in the number of self-organization actions. In this work 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 constraints to the resources of agents. Moreover, we plan to consider other self-organization actions such as organizational roles changes in the system (i.e., agents that acquire most demanded roles instead of leaving the system) in order to adapt their functionality to the service demand.

References

[1] S. Abdallah, V. Lesser, Multiagent reinforcement learning and self-organization in a network of agents, in: Proceedings of the Sixth Interna-

tional Joint Conference on Autonomous Agents and Multi-Agent Systems, IFAAMAS, 2007, pp. 172–179.

[2] Adamic, Zipf's law and the internet, Glottometrics 3 (2002) 143–150.

[3] M. Aquin, S. Elahi, E. Motta, Personal monitoring of web information exchange: Towards web lifelogging, in: Proceedings of the WebSci10, Raleigh, NC: US, 2010, pp. 1–7.

[4] B. Biskupski, J. Dowling, J. Sacha, Properties and mechanisms of self-organizing manet and p2p systems, ACM Trans. Auton. Adapt. Syst. 2 (2007) 1–34.

[5] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, D.U. Hwang, Complex networks: Structure and dynamics, Physics Reports 424 (2006) 175 – 308.

[6] M. Bowling, M. Veloso, Multiagent learning using a variable learning rate, Artificial Intelligence 136 (2002) 215–250.

[7] R. Buyya, Market-oriented cloud computing: Vision, hype, and reality of delivering computing as the 5th utility, in: Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, IEEE Computer Society, 2009, pp. 5 –13.

[8] B.F. Cooper, H. Garcia-Molina, Ad hoc, self-supervising peer-to-peer search networks, ACM Trans. Inf. Syst. 23 (2005) 169–200.

[9] A. Crespo, H. Garcia-Molina, Routing Indices For Peer-to-Peer Systems, in: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02), IEEE Computer Society, 2002, p. 23.

[10] E. Del Val, M. Rebollo, V. Botti, Decentralized Service Management based on Homophily for Self-Adaptive SOMAS, in: IEEE 8th International Conference on Services Computing, IEEE Computer Society, 2011, pp. 755–756.

[11] E. Del Val, M. Rebollo, V. Botti, Enhancing Decentralized Service Discovery in Open Service-Oriented Multi-Agent Systems, Journal of Autonomous Agents and Multi-Agent Systems (2012) 1–30.

[12] G. Di Caro, F. Ducatelle, L.M. Gambardella, Anthocnet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks, European Transactions On Telecommunications 16 (2005) 443–455.

[13] G. Di Marzo Serugendo, M.P. Gleizes, A. Karageorgos, Self-organization in multi-agent systems, Knowl. Eng. Rev. (2005) 165–189.

[14] J. Gabbai, H. Yin, W. Wright, N. Allinson, Self-organization, emergence and multi-agent systems, in: International Conference on Neural Networks and Brain, volume 3, IEEE Computational Intelligence Society, 2005, pp. 1858–1863.

[15] M.E. Gaston, M. DesJardins, Agent-organized networks for multi-agent production and exchange, in: Proceedings of the 20th National Conference on Artificial Intelligence - Volume 1, AAAI Press, 2005, pp. 77–82.

[16] A. Giagkos, M.S. Wilson, Beeip a swarm intelligence based routing for wireless ad hoc networks, Information Sciences 265 (2014) 23 – 35.

[17] C. Gkantsidis, M. Mihail, A. Saberi, Random walks in peer-to-peer networks: Algorithms and evaluation, Performance Evaluation 63 (2006) 241–263.

[18] P. Haase, R. Siebes, F. van Harmelen, Peer selection in peer-to-peer networks with semantic topologies, in: Proceedings of the International Conference on Semantics in a Networked World (ICNSW'04), volume 3226 of *LNCS*, Springer Verlag, 2004, pp. 108–125.

[19] P.N. Howard, L. Rainee, S. Jones, Days and nights on the internet, American Behavioural Scientist 45 (2001) 383–404.

[20] B.A. Huberman, L.A. Adamic, The Nature of Markets in the WWW, Technical Report, 1999.

[21] E. Kaddoum, C. Raibulet, J.P. Georgé, G. Picard, M.P. Gleizes, Criteria for the evaluation of self-* systems, in: Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, ACM Press, 2010, pp. 29–38.

[22] S. Kamboj, K.S. Decker, Organizational self-design in semi-dynamic environments, in: Proceedings of the 6th International Conference on Autonomous Agents and Multiagent Systems, ACM, 2007, pp. 335 – 337.

[23] J. Kleinberg, Complex networks and decentralized search algorithms, in: Proceedings of the International Congress of Mathematicians (ICM), European Mathematical Society, 2006, pp. 1–26.

[24] R. Kota, N. Gibbins, N.R. Jennings, Self-organising agent organisations, in: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, IFAAMAS, 2009, pp. 797–804.

[25] R. Kota, N. Gibbins, N.R. Jennings, Decentralized approaches for self-adaptation in agent organizations, ACM Trans. Auton. Adapt. Syst. 7 (2012) 1:1–1:28.

[26] P.F. Lazarsfeld, R.K. Merton, Friendship as a social process: A substantive and methodological analysis, Freedom and Control in Modern Society (1954) 18–66.

[27] P. Leito, Towards self-organized service-oriented multi-agent systems, in: Service Orientation in Holonic and Multi Agent Manufacturing and Robotics, volume 472 of *Studies in Computational Intelligence*, Springer Berlin Heidelberg, 2013, pp. 41–56.

[28] A. Loser, S. Staab, C. Tempich, Semantic social overlay networks, Selected Areas in Communications, IEEE Journal on 25 (2007) 5–14.

[29] J.P. Mano, C. Bourjot, G.A. Lopardo, P. Glize, Bio-inspired mechanisms for artificial self-organised systems, Informatica (Slovenia) (2006) 55–62.

[30] P. Maymounkov, D. Mazières, Kademlia: A peer-to-peer information system based on the xor metric, in: Revised Papers from the First International Workshop on Peer-to-Peer Systems, IPTPS '01, Springer-Verlag, London, UK, UK, 2002, pp. 53–65.

[31] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, A. Lser, Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks, in: Proceedings of the 12th International World Wide Web Conference, 2003, pp. 536–543.

[32] M.P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, Service-oriented computing: State of the art and research challenges, Computer 40 (2007) 38–45.

[33] M.P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, Service-Oriented computing: A research roadmap, International Journal of Cooperative Information Systems 17 (2008) 223–255.

[34] L. Peshkin, V. Savova, Reinforcement learning for adaptive routing, in: Proceedings of the International Joint Conference on Neural Networks, volume 2, IEEE Computational Intelligence Society, 2002, pp. 1825 –11830.

[35] C. Raibulet, L. Masciadri, Evaluation of dynamic adaptivity through metrics: an achievable target?, in: Proceedings of the European Conference on Software Architecture, IEEE, 2009, pp. 341 –3344.

[36] M. Ripeanu, Peer-to-peer architecture case study: Gnutella network, P2P (2001).

[37] R. Rodrigues, P. Druschel, Peer-to-peer systems, Commun. ACM 53 (2010) 72–82.

[38] M. Saleem, G.A.D. Caro, M. Farooq, Swarm intelligence based routing protocol for wireless sensor networks: Survey and future directions, Information Sciences 181 (2011) 4597 – 4624.

[39] Ö. Simsek, D. Jensen, Decentralized search in networks using homophily and degree disparity, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence, AAAI Press/International Joint Conferences on Artificial Intelligence, 2005, pp. 304–310.

[40] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications, Computer Communication Review 31 (2001) 149–160.

[41] C. Tempich, S. Staab, A. Wranik, REMINDIN': Semantic Query Routing in Peer to Peer Networks Based on Social Metaphors, in: Proceedings of the WWW2004, pp. 640–649.

[42] P. Velagapudi, O. Prokopyev, K. Sycara, P. Scerri, Analyzing the performance of randomized information sharing, in: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2, AAMAS '09, IFAAMAS, 2009, pp. 821–828.

[43] G.A. Vouros, Information searching and sharing in large-scale dynamic networks, in: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '07, ACM, 2007, pp. 49:1–49:8.

[44] D. Watts, P. Dodds, M. Newman, Identity and search in social networks, Science 296 (2002) 1302 – 1305.

[45] D. Weyns, M. Georgeff, Self-adaptation using multiagent systems, IEEE Software 27 (2010) 86 –891.

[46] Y. Xu, M. Lewis, K. Sycara, P. Scerri, Information sharing in large scale teams, in: AAMAS04 Workshop on challenges in coordination of large scale multiagent systems.

[47] B. Yang, H. Garcia-Molina, Efficient search in peer-to-peer networks, in: Proceedings of the International Conference on Distributed Computing Systems (ICDCS), 2002.

[48] D. Ye, M. Zhang, D. Sutanto, Self-organization in an agent network: A mechanism and a potential application, Decision Support Systems 53 (2012) 406 – 417.

[49] B. Yu, M.P. Singh, Searching social networks, in: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '03, ACM, 2003, pp. 65–72.

[50] H. Zhang, B. Croft, B. Levine, V. Lesser, A Multi-agent Approach for Peer-to-Peer based Information Retrieval System, in: Proceedings of 3rd International Joint Conference on Autonomous Agents and MultiAgent Systems, 2004, pp. 456–464.

[51] H. Zhang, V. Lesser, Multi-Agent Based Peer-to-Peer Information Retrieval Systems With Concurrent Search Sessions, in: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, ACM, 2006, pp. 305–312.