Dynamic Composition of Service Oriented Multi-agent System in Self-organized Environments

Nelson Rodrigues[1,3] and **Paulo Leitão**[1,3] and **Eugénio Oliveira**[2,3]

Abstract. The increasing relevance of complex systems in dynamic environments has received special attention from researchers during the last decade. Due to the need of a flexible and quick response to the clients' requirements, such systems become an important challenge. In this paper, self-organizing mechanisms capable to compose services in an automatic, flexible and decentralized manner are presented, mostly in which their adaptive behavior is concerned. Due to the distributed approach, we also investigate the adaptation regarding the structure of each entity. We thus propose an innovative self-learning mechanism that allows the distributed entities to learn structural relations allowing the system's evolution. This hypothesis were explored and validated by implementing a multi-agent system, in accordance with trust mechanisms to improve the interaction of agents.

The achieved results show the correct agent's states in which the agents must evolve and self-organize, improving the system benefits band increasing the organization performance.

1 INTRODUCTION

The increasing relevance of complex systems in dynamic environments (e.g. the buyer-supplier network) has received special attention during the last decade from the researchers. Such systems need to satisfy client's desires, which, after being accomplished might change again, thus becoming a very dynamic situation [1]. Usually centralized approaches are implemented, which might fall into a large monolithic software packages, being inadequate because they do not efficiently support the needed flexibility and real time re-configurability. Concentrating the entire control on a single coordinator can create a bottleneck and, also, the coordinator needs to have previous knowledge about each web service component in the environment. Currently, based on the benefits of distributed control, decentralized approaches have been pointed out as fitted to address this challenge [2]. On the other hand, the implementation of decentralized discovery, composition and execution can also increase the complexity of the system regarding the network traffic to properly coordinate as the distributed control itself [3]. Thus, adaptation and intelligence, considering the requirements, must be properly addressed, being multi-agent systems (MAS) a suitable paradigm for supporting such distributed intelligence. A flexible and automatic integration can be achieved by joining the intelligence and autonomy provided by multi-agent systems and the interoperability offered by Service-Oriented Architecture (SOA) solutions [4].

This work explores the service-oriented multi-agent system benefits together with self-organization principles. Distributed self-adaptation changes in each entity allow agents to fulfil the client's needs, by providing agility and quick responses, permitting to the clients to save time and money by simplifying the task's complexity to be carried out. To accomplish the client's needs it is necessary, in a dynamic manner, to compose a set of services provided by multi-agent systems, in order to offer better solutions to the clients. Clearly, the minimization of the planning and execution time of services on demand, must be considered. Particularly, and since the self-organization principle is explored, it is necessary to take into consideration the Quality of Services (QoS) [5] of composition as a continuous task to carry on.

QoS have to be based on service performance, cost, availability, response time and also the trustworthiness of the agent. It will be possible to adapt the agent's behaviors selection that will indirectly change the network topology to a more consistent one, structured with higher quality and better trustworthiness of the performed services. The assumptions on this approach allow joining several agents belonging to different societies after they make their mutual connections evolve. Due to the automatic creation of new relations, an agent can smoothly enter into a society. On the opposite side, the weak connections might be eliminated avoiding saturation of the system with useless agents. In this way, the system achieves equilibrium regarding the responsiveness from all societies, since the agents are able to evolve to accomplish the requests.

The experimental results we have obtained highlight the benefits of a truly distributed and decentralized solution that performs an accurate self-organization model, which directly impacts the system performance. The system is able to self-adapt by performing the same services according to the client's needs, but with less costs and better response times.

The rest of the paper is organized as follows: Section 2 discusses the related work and Section 3 overviews the main principles of the self-organized, service-oriented, agent-based architecture, for dynamic and reconfigurable systems. Section 4 presents the formalization of the proposed model for the dynamic, decentralized service composition in evolutionary systems. Section

[1] LIACC - Artificial Intelligence and Computer Science Laboratory, Rua Campo Alegre 102, 4169-007 Porto, Portugal
[2] Faculty of Engineering - University of Porto, Rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal , email: eco@fe.up.pt
[3] Polytechnic Institute of Bragança, Campus Sta Apolónia, Apartado 1134, 5301-857 Bragança, email: {nrodrigues, pleitao}@ipb.pt

IAT4SIS '14, August 18 2014, Prague, Czech Republic

http://dx.doi.org/10.1145/2655985.2655990

5 details the system evolution of the learning module embedded in each agent that will support the service discovery and composition phases. Section 6 describes the experimental setup and analyses the achieved results. Finally, Section 7 wraps up the work done with the conclusions and future work.

2 RELATED WORK

Self-organization was originally introduced by Ashby [5], and refers to a process of cooperation between individual entities without any centralized decision or centralized knowledge. In dynamic and complex systems, it arises as a coordinated global model [6] of a cooperative chaos. Self-organization systems are usually studied in the economy, biology, computer science and other fields [7], [8]. Several authors have proposed different representations of self-organization models, for example, cellular automata and differential equations, fail to move to the realistic prototype, in a simple manner [9]. The literature review addresses a large variety of self-organized systems, given some importance to the service selection, which has been widely studied. Unfortunately, the majority of the approaches for service discovery process rely on a centralized repository and therefore the self-organization of the system is centered in a bottleneck issue. Several attempts have been tried to understand the benefits of the distributed adaptations, for example based on biological cells to maintain an efficient and robust network [10]. Clearly, the key question is "when and how do adapt" aiming to have a dynamic and adaptable network.

The interaction among the entities or observations is the answer for the "how". For example, Val et al. [11] propose a self-organization network, where the agent's relations are created based on the social plasticity and incentives discovered on the network. In relation to "when", some approaches [11] change the system due to new policies and requirements from the consumer [6], when a new service is requested [12] or in the worst cases when an error or disturbance occurs. For instance, Vogel and Giese [13] take into consideration the feedback loops and the criteria of QoS, using a model-driven approach for self-adaptation, referred as off-line adaptation. Our approach uses an agent-based model, since it is particularly easy and flexible to model complex systems as a collection of cooperative autonomous agents. Our proposal differs from the previous, since it evolves on the fly [1] without the need to stop, reprogram and reinitialize [14], [15], [11], through the local behavior and structural relations adaptation. Furthermore, learning mechanisms are demonstrated to maximize the confidence of the evolution, particularly from whom and when to evolve.

3 SERVICE-ORIENTED AGENT-BASED SYSTEMS

The proposed architecture, combines service-oriented and multi-agent systems paradigms, taking advantage of some points previously investigated [1]. The first paradigm represents the interoperability, loose coupling and an abstraction of the business logic. The second one provides decentralization, distribution of decision-making entities and autonomy. Therefore, it arises the necessity to separate a set of intelligent and autonomous agents, capable to cooperate in order to accomplish the client's requirements from the static part represented by the services, as illustrated in Fig. 1.

In the design agent's societies, the following agent's roles were defined: *consumer*, *provider*, *workflow*, *ontology* and *reputation*. A *consumer* represents the role entity that has the need to request a service, a *provider* has the capability to provide one or more services that correspond to its skills. Note that each agent has the autonomy to choose the service that it wants to offer, as well the conditions, e.g. price and QoS.

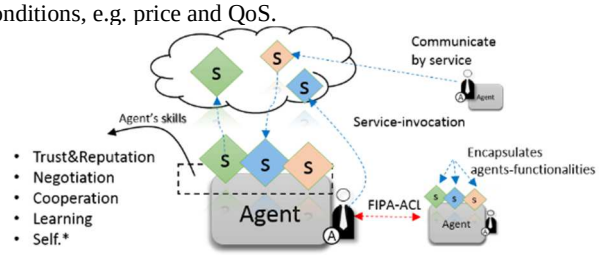

Figure 1. Combining the interoperability of SOA and the intelligence and autonomy of MAS [1].

Additionally, due to the provider due to the provider role, the agent can decide according to its skills which service becomes public, in order to be offered. The *workflow* monitors QoS and trust values of available services, to perform, in a dynamic way, the proposed workflow composition based on several criteria, e.g. availability and price. The *reputation* gathers agent's opinions about a specific service for future advices. Lastly, the agents have the semantic capability, by the *ontology* role, to translate concepts, in order to support the proper understanding among the agents and to solve misunderstandings based on similarity mechanisms. Thus the agents proposes, continuously, a set of hypotheses of the complex task of service composition. The proposed architecture follows a truly decentralized system, without the presence of a central node that supports the discovery of the desired services. This means the removal of centralized coordinator, which can be a probable bottleneck.

4 DYNAMIC AND DECENTRALIZED SERVICE COMPOSITION

In distributed systems, agents must cooperate among them themselves to perform their goals and increase the utility of the whole systems. The self-organization of the services provided by the agent-based system, appears as a crucial role in such environments. In our approach the self-organization of services comprises the execution of the following tasks, discovery, in a decentralized manner, the desired dynamic composition of services, and finally execute the best designed service composition. Algorithm 1 describes the decentralized discovery and selection of the most trusted neighboring agents to achieve the composition.

4.1 Decentralized discovery

The discovery procedure is triggered by an agent, when it receives a request to provide the service *agInput* (*line 1*), which is not able to perform (*line 20*). In this situation, or when the composition quality is too low, the agent tries to find potential neighbors who can offer the service (*lines 9, 23*), propagating the service request. The discovery phase gathers only most promising agents [15] to execute the service or to offer their atomic services (*lines 7, 21*). It is necessary a multi-criteria function, to select a possible agent, formalized by a Multi-Attribute Utility Theory (MAUT) to maximize the utility value, represented as, $U = \sum(\omega_i * trust) + (\omega_i * QoS)$.

Algorithm 1: ***Requests*** *(agInput)*

```
1.  compResult ← CompositionStrips(agInput)
2.  if compResult = successful then
3.    conf ← calculateQoSTrust(compResult)
4.    if conf > 0.5 then           //0.5 for example
5.       results ← execCompOperation (compResult)
6.    else
7.       Nt ← SelectNeighbor(Aginput, this)
8.       TTL ← 9                  //for example
9.       Results ← forwardMessage(Aginput, TTL, Nt)
10.      Sol ← rankSolution(Results)
11.      compResAux ← CompositionStrips(Sol)
12.      confAux ← calculateQoSTrust(compResAux)
13.      if(confAux >conf & compResAux =successful) then
14.         results = execCompOperation(compResAux)
15.      else
16.         results = execCompOperation(compResult)
17.      endIf
18.   endIf
19.   updateAnalyze(this, agInput, results)
20. else
21.   Nt ← SelectNeighbor(Aginput, this)
22.   While Nt.size <> 0 do
23.      Results←forwardMessage(Aginput,TTL, nt(i))
24.   endWhile
25.   Sol ← rankSolution(Results)
26.   Requests(sol, Aginput) //forward the request soution
27. endIf
```

In this multi-attribute function, the utility U stands for the overall expected utility of different criteria, namely trust, reputation, QoS and price, being ω_i the weight of the criteria.

In order to control the amount of messages exchanged, in larger societies, it is defined a cost to manage how deep the message can go, namely how many times they are forwarded (*line 8*). The agent after spreading the requests, waits for the filtered comeback responses, during a specific time, then the agent must select according to the consumer's requirements the most appropriate service composition to execute (*lines 5, 14, 16)*.

4.2 Service composition

The service composition, considering the set of services discovered in the previous phase is implemented in a decentralized manner. The service composition takes insights from the services' choreography, which defines the way the services are connected and the data flows in specific directions [16]. Particularly, each agent can produce various plans with the same set of services, creating several composition hypotheses for the same input requirements (*lines 1, 11*).

The planning algorithm embedded in each agent for the service composition is based on the well-known automated STRIPS planners, which input information comprises, mainly, the pre- and post-conditions for each action and the goal that the algorithm is trying to reach [17].

The results of the algorithm are a set of possible service compositions, the agent selects the plan (composition) with the best quality. Particularly, it is necessary another multi-criteria function, to select the best composition, based on global QoS, cost and price.

If the agent needs to ask for distributed composition help (*lines 9, 23*), it is necessary to calculate the overall composition produced, by collecting from each composition participant the service quality, price and trust. Being up to the consumer to take the decision to accept or not such composition.

5 SYSTEM EVOLUTION

During the agent life-cycle, the agent is continuously trying to learn from its history. The historical information, collected from the learning agent's interactions, permits the agents to evolve more accurately. The agents can adapt and evolve in terms of behavioral self-organization, which means, modifications from the agent's local behavior, e.g. learn new services by analogy or adapt the execution parameters, and secondly of structural self-organization, for example, modification of the relationships among agents.

5.1 Learning Behavioral Self-organization

The execution of behavioral adaptation will be supported by embedding learning mechanisms. The agents can adapt their knowledge, regarding the behavior, immediately after executing and updating the execution (*line 19* from Algorithm 1). Algorithm 2 describes the agent collecting information from the environment. In our case, this information is gathered from the interactions of agents, in which it is possible to qualify the action performed.

In an effort to take accurate and better decisions in the future, allowing the self-organization of future behaviors, the agent updates and analyses its memory according to the execution results contracted (*lines 4 to 8*), these lines measure the satisfaction of the contract compared to what really happened. For example, QoS and trust values after being saved and aggregated on the database (*lines 1, 2)*, are calculated the actual values for the particular agent (*lines 3, 4*), being these values used later.

All these steps lead to the best trustworthy choice in the future allowing the behavioral adaptation. However, this adaption is a continuous task, since the feedback to assign agreements will change, in a distributed manner, for example, in a particular context, an agent can offer services with high confidence, which is variable over time.

Algorithm 2: ***updateAnalyze*** *(agent, service, results)*

```
1. updateQoS(agent, service, results)
2. updateTrust(agent, service, results)
3. trustValue ← getTrustResult(agent, service,results)
4. QoSValue ← getQoSResult (agent, results)
5. reward ← computeReward(trustValue, QoSValue)
6. state ← getAgentState (agent)
7. act ← getAgentAction (agent)
8. updateReinforcementLearning(state, act, reward)
9. createConnection(state, act, reward)
```

Just as important as the behavioral adaptation is the structural adaptation in the evolution of the system. The last line of Algorithm 2 considers the evolution in the network, explicitly the structural adaptation by managing the structural relations. In addition, a novel approach it is introduced, which advocates that, at least the structural decisions of the agents should be performed when the knowledge is stable.

5.2 Learning mechanism to support the Structural Self-Organization

The agent possesses the necessary information to recognize if it is capable to evolve its structural relations in a confident manner or not, realizing if there is a slight disturbance that might produce significant results, leading to adapt to become more stable, and also to ensure that the system becomes steadier and consequently more robust to failure events.

This kind of knowledge is taken based on the agent's learning mechanism, which recognizes the reward value (positive or negative) of a specific action, in an agent's internal state. Thus the Q-learning algorithm was used [11], since it is a well-known reinforcement algorithm that allows getting the expected utility of

actions at particular states without the need to know the environment model, actually the environment model is built after several interactions. In this specific case the agent only needs to recognize the reward for a specific action in a particular state. The value associated to each state-action pair, namely the *q-value*, represents the expected accumulated reward for a particular action at a given state. In our particular case the state is defined in the expression (1).

$$\text{State} = < \text{ActualTrust}, \text{ActualQoS}, \text{Si} > \quad (1)$$

$$\text{Si} \in \{Sv1, \ldots, Svn\}$$
$$\text{actualTrust} \in \{good, normal, bad\}$$
$$actualQoS \in \{good, normal, bad\}$$

The state represents the agent's state at a given instant. This level of trust is built by the state and the quality of a particular service. The representation of an action is presented in expression (2),

$$\text{Action} = < \text{path}, \text{Si} > \quad (2)$$

$$path \in \{\text{Ag}_1, \ldots, \text{Ag}_n\}$$

This second q-learning vector is built by the set of agents that will create a path of agents that are necessary to execute the service. The reward considers the agent feedback state-action pair, and its result is defined in the expression (3),

$$\text{Reward} = <\text{State}, \text{Action}, \text{AgentFeedback}> \quad (3)$$

The reward is calculated by a multi-attribute function, equation (4), after the agent has executed the service, using a given weight for each feedback, results in a single value, where x represents (QoSFeedback × w1 + trustFeedback × w2), note that the range values $\{ value1, value2, value3\}$, can be parameterized, but we will use $\{ value1 = 0.5, value2 = 0.4, value3 = 0.1\}$

$$Maut\,(x) = \begin{cases} if\ value1 \geq result \leq 1, & good \\ if\ value2\ = result, & normal \\ if\ value3 \geq result \leq value2, & bad \end{cases} \quad (4)$$

After getting the classification of {good, normal, bad} the reward analysis is performed. If it is considered good, then a reward r is given; if the result is "normal", then the reward is half of the "good" reward, since it has finished the task; If it is "bad" it suffers a penalization. Summing up, the reward function will retrieve a quantitative value from {good, normal, bad}, see equation (5).

$$Reward(result) = \begin{cases} if\ good\,, & r \\ if\ normal, & r/2 \\ if\ bad, & \frac{r}{2} * penalization \end{cases} \quad (5)$$

Thus, reward and punishment policies can be modeled in order to manage the responsibility of an agreed result.

Taking into account the learning model, the system can evolve in a stable manner, i.e. when it is converging (see Fig 2). If such behavior occurs, the agent proposes the creation of new connections.

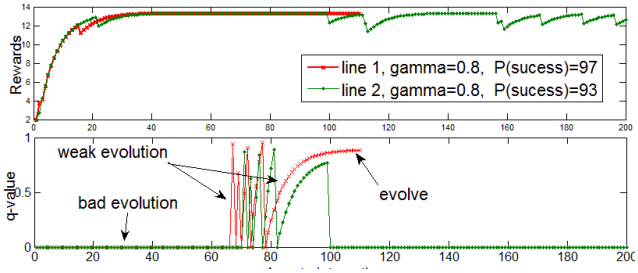

Figure 2. Learning mechanism for a specific service, contrasting the value of learning with particular reward.

Analyzing the *q-value* axis in Fig. 2 it is notorious a tendency of the line 1, with the probability of success agreement between two agents of 97%, converging faster. In the converging phase, for example, at iteration 110, the agent is in a good position to evolve its structural relations with more utility. If the agent evolves too quickly, for example in iteration 70, the utility will be worse, since the agent is not stable to take an accurate decision.

The utility of the structural self-organization is, therefore, strongly related to the success of the stability, for example the line 2, with P(success)=93%, will take more time to become stable, since it has more broken contracts than with higher probability, and thus a more inaccurate behavior.

After the agent realizes that when it is a good time to evolve structurally, it efforts go into its structural model adaptation, trying to optimize its own relations, for instance to cooperate directly with other agents, and thus create new connections more valuable. In Algorithm 3 the function to manage the creation of the new connection is defined.

Algorithm 3: ***createConnection*** *(state, action, reward)*

```
1.  Q'← learning(state, action, reward)
2.  convergenceValue ← 0.001
3.  if (oldLearning-Q') < convergenceValue then
4.    if count > countStability then //100 for example
5.      agent ← adaptModel()
6.      sendMessage(agent)
7.    else
8.      count ← count + 1
9.    endIf
10. else
11.   count ← 0
12. endIf
```

Analyzing the algorithm, each agent compares its own old knowledge base with the latest one (*line 3*), if the difference is less than convergence value, then it is considered a possible stable system, although it is necessary to have this same result during some amount of others iterations, in order to justify the stability (*line 4*). When the agent believes that it has a stable behavior, he tries to adapt the model of their world (*line 5*). Spontaneously, in order to give opportunities to cooperate with new agents, some of the time it is selects an agent with lower quality and trust value, only if the reputation of that specific agent is not negative (*line 6*), thus allowing the analyses of exploitation vs. exploration of the system.

The proposed structural model does not stop the agent's behaviors, since it is triggered as parallel behavior, thus it continues to answer to other requests.

6 EXPERIMENTAL SETUP/VALIDATION

The proposed self-organization model was implemented and experimentally tested in a case study where evolution is a requirement.

6.1 Experimental setup

In the implemented setup, a society of agents following the described approach was implemented by using the JADE framework [18]. Each agent includes the roles of *producer*, and *consumer*. Thus it was created a network with 100 agents with random connections among them, to each agent it was assigned a set of services. Based on the assumption that different agents can produce the same services, but with different costs, thus it were defined the cost of the services by each agent (besides of the

service execution cost, the agents must pay for forwarding messages, this way we can control the amount of forwarded messages), it also represents the probabilities of success for agreements fulfilled, being these values created randomly in the agent network creation phase. Then, several service requests to different agents were performed, allowing the composition, discovery and execution tasks to be computed automatically, expecting in parallel, the self-organization phenomenon.

The implementation considers a system of decentralized discovery, where is necessary to cooperate, creating a primitive *frame (Id, TTL, Src, Dst, Sv, Perf, Res)* to send between neighbors. The *Id* reflects the identification of the service (*Sv*) request. *Src* tag, represents the agent source and the *Dst* the agent destination. The *Perf* indicates the type to request {*Request, Propose, Execute, etc.*} that will be forwarded until the TTL reaches 0, and finally the *Res* is a container that comprises the result information of that frame.

Considering the structural learning mechanism, a random walk algorithm was implemented, in order to select the entry point of the service request, avoiding a greedy selection by requesting always to the same agent. The Q-learning configuration in each agent corresponds to gamma=0.8, the r=1 for the good, normal or bad rewards, and the acceptance of a good learning convergence is equal or up to th=0.5.

6.2 Self-organization results

All the agents of the generated society can receive service requests and answer them in the best possible manner, by trying to compose, discover and select the most appropriate agents to offer their services. To illustrate the proposed benefits, it was extracted from the scenario produced a network of agents. Each service is composed by de-coupling of other services, namely, ServiceA={s_1,s_2,s_3} and ServiceB={s_3,s_4,s_5}. It can be observed what services are provided by each agent, for example agent *agB* offers the services {s_2, s_5} at the prices 1, 2 respectively. The probability for the success agreement contract, with agent agC as shown is 80%. It is notorious how the discovery flows in the network. Requests are directional, and responses are in counter-flow, for instance, agent agA can make requests to agent agB, after that, agent agB to agent agC, but then, the agent agC cannot forward requests to its neighbors.

The goal, as already stated, is to create automatically the best composition, forwarding messages if needed. Taking into consideration the global costs, which can be allocated to messages' average cost before and after the evolution, as well as the services' execution costs, which are stored in a vector apart, allowing a clearer analytical perception. The insights about this work demonstrate a network capable to solve the requests with less agents, in the same dataset.

Fig. 3 describes the relationships in the network between agents, before evolving (depicted by arrows with normal line). The dotted lines illustrate the potential connections that the agent agE proposes, according to their experience. The agent agA have the link to the agent agB, however, after running the system requesting the ServiceA, the agent agE will learn the environment connections, and then proposes a new cooperative link {agE→ agB}.

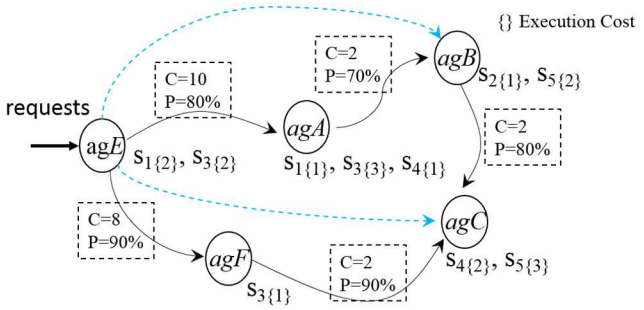

Figure 3. Agent's network from, the agent agE point of view.

At this point the agent sends a message to cooperate directly with the agent agB, in order to reduce the execution costs. The probabilities, created for the new links, as well as the costs are created randomly. Obviously new connections with higher costs and lower success probabilities, are not created. The agent agE executes the service composition with lower cost, defining cost as $Cost=w_1\times \square_i + w_2\times\lozenge_i + w_3\times(1-\circ_i)$, the tests were carried out for the entire network with the weights w_1 =0.2, w_2=0.3 and w_3=0.5;

Table 1. Costs involved before and after the evolution occurred for the request of the ServiceA.

Agent's chain	□	◊	○	Cost
Before Evolution				
agE(s_1)→agA(s_3)→agB(s_2)	12	6	0.56	4.42
agE(s1)→agE(s3)→agA→agB(s2)	12	5	0.56	4.12
agE→agA(s1)→agA(s3)→agB(s2)	12	5	0.56	4.12
After Evolution				
agE(s1)→agE(s3)→agB(s2)	2	5	80	2

◊ execution cost □ connection cost ○ probability success

Regarding the execution's results of the ServiceA illustrated in Table 1, the agent agE can evolve. Considering one composition, for example agent agE forwards the request to agent agA, then agent agA can execute the service s1, and service s3, but it is still missing service s3, which will be executed by agent agB by the forwarding request of agent agA. The following procedure can be stated as agE → agA(s1) → agA(s3) → agB(s2), where:

- execution cost is calculated by the table in Fig. 3. Execution of service s1 and s3 by the agent agA it costs 1 and 3 respectively and the service s2 in agent agB costs 1 (◊ = 5).
- connection cost considers the connections from agE to agA to agB (□ = 12).
- regarding the probability results, it is considered the probability of agE with agA, and agA with agB (○ = 0.80 × 0.70 = 0.56), see the graph in Fig 3. for further details.

Since the q-value from the agent agE has stabilized, then the agent begins to explore the ServiceA composition, using the agE(s1) → agB(s2) → agE(s3) composition. Since this composition has a low cost of execution and connection, and also a connection with greater confidence, thereby being selected for future requests.

For the execution of ServiceB, agent agE is also required as an entry point. After several requests the agent agE proposes a new connection agE → agC, with probability success 70%, see Table 2. After the evolution of these two connections have occurred, agent agE when queried to perform ServiceB, it tends for the following selection agE(s3)→agB(s5)→agC(s4), because of the cost of the composition. Note that the agents can explore all possible compositions that are offered by the planner, for example the agent agE when performs the serviceB take advantages of connection created regarding the execution of serviceA.

Table 2. Costs involved before and after the evolution occurred for the request of the ServiceB.

Agent's chain	□	◊	○	Cost
Before Evolution				
agE(s3)→agF→agC(s4)→agC(s5)	10	7	0.81	4.195
agE→agF(s3)→agC(s4)→agC(s5)	10	6	0.81	3.895
agE→agA(s3)→agB→agC(s4)→agC(s5)	14	8	0.448	5.476
agE(s3)→agA→agB→agC(s4)→agC(s5)	14	7	0.448	5.176
agE(s3)→agA(s4)→agB→agC(s5)	14	3	0.448	3.976
agE(s3)→agA→agB(s5)→agC(s4)	14	6	0.448	4.876
agE(s3)→agA(s4)→agB(s5)	12	5	0.56	4.12
After Evolution				
agE(s3)→agC(s4)→agC(s5)	4	7	0.7	3.05
agE(s3)→agB(s5)→agC(s4)	4	6	0.64	2.78
agE(s3)→agB→agC(s4) →agC(s5)	4	7	0.64	3.08

◊ execution cost □ connection cost ○ probability success

Therefore, it can be concluded that all links created bring benefits, otherwise the agents will not reach to an agreement and they will not cooperate. However, this does not mean that agents will use it permanently. Note that the goal is cooperating and evolve in order to reduce the service composition costs, by exploring different agents, to be offered to the client.

The system created was tested following the principles based on the self-organization paradigms [8], namely multiple interactions from the multi-agent system. We have tested also, the benefits of the system stability in terms of exploitation and exploration of new relations achieved through the positives and negatives rewards.

7 CONCLUSIONS

This paper starts by referring the research efforts done on service composition in complex systems, where all network control is centralized in a single entity. Some proposed solutions, consider centralized service registrations, which leads to a bottleneck-type of problem. Taking this issue into consideration, and expecting to provide services solutions with lowest throughput costs, we introduce a truly distributed and decentralized service oriented agent-based system that positively answers this challenge. Despite the fact that decentralization brings performance benefits, it also increases the complexity, as well as the data flow to keep the distribution synchronized and coordinated. Thus, it requires behavioral and structural modifications to the system, in order to enhance the decentralization benefits. We advocate that a dynamic system capable to discover, compose through interactions and negotiation protocols, provides an answer to the service requested to be solved in a self-organized manner. As a consequence, there is a reduction of the unnecessary traffic in the network to compose or execute a service by evolving in a correct way.

The developed scenario concerns several service requests to the society of agents, which comprise a set of autonomous service oriented agents with no global control. The implementation takes into account a decentralized discovery mechanism to search in a dynamic manner for services that are then composed considering their quality and trustworthiness. The proposed solution allows the agents to adapt its structural relations, based on the reinforcement learning mechanism. The experiment allows to execute different flows of services composition, fulfilling every time the client needs, namely the continuous requests to the system. Additionally the system's organization performance has improved, due to the self-organization achieved through the learning mechanisms and interactions among the distributed agents. As a result, more trusted services with higher quality are allowed.

As a future work, we intend to investigate the design and implementation of our approach in a real case scenario, for example smart grids or business to business network operations. Also explore scenarios of self-organization where the intermediate agents of service requests, do not share information about their neighbors, exploring the benefits of selfish agent providers.

REFERENCES

[1] N. Rodrigues, E. Oliveira, and P. Leitão, "Self-organization Combining Incentives and Risk Management for a Dynamic Service-Oriented Multi-agent System," in Technological Innovation for Collective Awareness Systems, Springer Berlin Heidelberg, pp. 101–108, 2014.

[2] M. J. Woolridge, Introduction to Multiagent Systems. New York, NY, USA: John Wiley and; Sons, Inc., 2002.

[3] G. B. Chafle, S. Chandra, V. Mann, and M. G. Nanda, "Decentralized Orchestration of Composite Web Services," in Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers &Amp; Posters, pp. 134–143, 2004.

[4] M. N. Huhns, "Agents as Web services," Internet Comput. IEEE, vol. 6, no. 4, pp. 93–95, 2002.

[5] W. R. Ashby, "Principles of the self-organizing dynamic system," J. Gen. Psychol., vol. 37, pp. 125–128, 1947.

[6] Y. Bar-Yam, "Dynamics of complex systems," Computers in Physics, vol. 12. p. 864, 2003.

[7] F. Heylighen, "The science of self-organization and adaptivity," Encycl. Life Support Syst., vol. 5, no. 3, pp. 253–280, 2001.

[8] E. Bonabeau, M. Dorigo, and G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems. New York, NY, USA: Oxford University Press, Inc., 1999.

[9] W. Elmenreich, R. D'Souza, C. Bettstetter, and H. Meer, "A Survey of Models and Design Methods for Self-organizing Networked Systems," in Proceedings of the 4th IFIP TC 6 International Workshop on Self-Organizing Systems, pp. 37–49, 2009.

[10] M. Beber and M.-T. Hütt, "How do production systems in biological cells maintain their function in changing environments?," Logist. Res., vol. 5, no. 3–4, pp. 79–87, 2012.

[11] E. del Val, M. Rebollo, and V. Botti, "Combination of self-organization mechanisms to enhance service discovery in open systems," Inf. Sci., 2014.

[12] E. M. Maximilien and M. P. Singh, "A Framework and Ontology for Dynamic Web Services Selection," IEEE Internet Comput., vol. 8, no. 5, pp. 84–93, 2004.

[13] T. Vogel and H. Giese, "Model-Driven Engineering of Self-Adaptive Software with EUREMA," ACM Trans. Auton. Adapt. Syst., vol. 8, pp. 1–33, 2014.

[14] L. N. Foner, "A multi-agent referral system for matchmaking," First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology. pp. 245–261, 1996.

[15] J. Urbano, A. Rocha, and E. Oliveira, "Trust Evaluation for Reliable Electronic Transactions between Business Partners," in Agent-Based Technologies and Applications for Enterprise Interoperability SE - 12, vol. 98, K. Fischer, J. Müller, and R. Levy, Eds. Springer Berlin Heidelberg, pp. 219–237, 2012.

[16] W3C Working Group, "Web Services Choreography Working Group." accessed on May, 1. 2014.

[17] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," Artificial Intelligence, vol. 2. pp. 189–208, 1971.

[18] F. L. Bellifemine, G. Caire, and D. Greenwood, Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology). John Wiley & Sons, 2007.