A Proximity-Based Self-Organizing Framework for Service Composition and Discovery

Agostino Forestiero, Carlo Mastroianni, Giuseppe Papuzzo, Giandomenico Spezzano
ICAR-CNR
87036 Rende (CS), Italy
{*forestiero,mastroianni,papuzzo,spezzano*}*@icar.cnr.it*

***Abstract*—The ICT market is experiencing an important shift from the request/provisioning of products toward a service-oriented view where everything (computing, storage, applications) is provided as a network-enabled service. It often happens that a solution to a problem cannot be offered by a single service, but by composing multiple basic services in a workflow. Service composition is indeed an important research topic that involves issues such as the design and execution of a workflow and the discovery of the component services on the network. This paper deals with the latter issue and presents an ant-inspired framework that facilitates collective discovery requests, issued to search a network for all the basic services that will compose a specific workflow. The idea is to reorganize the services so that the descriptors of services that are often used together are placed in neighbor peers. This helps a single query to find multiple basic services, which decreases the number of necessary queries and, consequently, lowers the search time and the network load.**

***Keywords*-bio-inspired; peer-to-peer; resource discovery; self-organization; service composition;**

I. Introduction

The market of information systems is experiencing an important shift from the request/provisioning of products toward a service-oriented view where everything (computing, storage, applications) is provided as a network-enabled service. This emerging service-centered paradigm, often referred to as the "Internet of Services" [1], aims at enabling a scalable architecture where software modules belonging to different domains can be accessed following a common paradigm and composed in multiple ways to meet the ever-changing requirements of today's business environments.

In this new setting, organizations are blurring their boundaries by interacting with one another and creating new business paradigms and organizational forms that transcend the previous static and closed competitive models. In this way, new value can be added to organizational services as they are placed in a wider context. The Internet of Services is expected to have a great impact not only in business scenarios, but also in advancing public administration procedures and services. This Internet of Services is supported by modern distributed technologies such as the Grid Computing [2] and the newly emerging Cloud Computing [3], with paradigms such as the "Software as a service" and the "Platform as a service".

Fundamental to the implementation of the Internet of Services is the effective management of the different phases of a service life cycle, including discovery, selection and composition. Service composition is a critical issue because, owing to the wide variety and evolving needs of users, a required service is not always readily available but may be created at design or at run time through the composition of pre-existent basic services [4] [5]. Unfortunately, as the number of available services increases, the number of composition possibilities grows exponentially. This is referred to as the problem of combinatorial explosion [1].

A great research effort is currently devoted to the building of automatic and semi-automatic frameworks and tools that assist the user in three main tasks: the design of complex services, or "workflows"; the discovery of the basic component services; the actual execution over the Internet of Services. The design phase is often assisted by tools that exploit statistics on the way services have been selected and composed in the past [6] and can use semantic and ontology-oriented algorithms. Once a composite service has been designed, the basic services specified in the composition pattern must be discovered on the network. In most cases, particularly in Grid and Cloud systems, it is not required to discover a specific service, but a number of services having desired characteristics, among which the user will select the most convenient at the execution time. Information about services is usually stored in *service descriptors* that are managed by centralized or distributed repositories. A discovery request must be issued to find these descriptors: in the case of a composite service, a request is necessary for each component/basic service, which can result in long overall discovery times and high network loads.

This paper presents a framework that clusters service descriptors spatially on the basis of the co-use frequencies of corresponding services. In other words, the descriptors of services that are often used together (i.e., in the same workflow) are placed in restricted regions of the network. The objective is to facilitate "collective" discovery services, which try to find all or most of the required basic services in a single shot, so as to save computational time, bandwidth, and reduce the response time experienced by the user.

The framework is decentralized and self-organizing, which guarantees good scalability and fault tolerance characteristics. These features are obtained by means of an *ant algorithm*. Ant algorithms are inspired by the behavior of some species of ants [7], and can be seen as a subclass of *swarm intelligence* algorithms, whose goal is to let complex and intelligent behavior emerge from simple operations performed by a large number of agents. In our framework, ant-inspired agents travel the network exploiting peer-to-peer (P2P) interconnections among hosts, and relocate the service descriptors through *pick* and *drop* operations driven by probabilistic choices. Service descriptors are ordered spatially so that frequently co-used descriptors can be easily found in neighbor hosts. This allows search messages to find the desired services by approaching the hosts that store the corresponding descriptors, hop by hop. Moreover, once a search message gets to the descriptor of a service, it will easily find other useful services in the same local region. Performance analysis based on event-based simulation shows that the ant algorithm succeeds in the spatial ordering of descriptors, and the associated discovery algorithm allows search requests to better satisfy user requirements with a lower resource consumption.

The rest of the paper is organized as follows: after the related work section, Section III describes the algorithms for the reorganization and discovery of service descriptors; Section IV presents performance evaluation results; Section V concludes the paper.

II. Related work

The ability to select and integrate inter-organizational and heterogeneous services is an important step towards the development of applications over service-oriented frameworks, such as Grids and Clouds. If no single service can satisfy the functionalities required by the user, it is often possible to combine existing services in order to fulfil the request. This trend has triggered a considerable number of research efforts on the composition of services in academia and in industry [8] [9] [10]. Despite all these efforts, service composition is still a highly complex task. The complexity comes from many sources: firstly, the number of available services increases dramatically and composition alternatives grow exponentially; secondly, services can be created and updated on the fly, thus the composition should be made based on ever updated information; thirdly, services can be developed by different organizations, which use different concept models to describe the services, and yet there is no unique language to define and evaluate the services.

Service composition is today typically performed through centralized middleware components, such as registries, discovery engines and brokers. This approach is inevitably a serious bottleneck if the number of services and organizations is large. Distributed registries and registry federations have been proposed as a first approach to avoid bottlenecks in a service network. Some existing standards, for example EbXML and UDDI, adopt a federation of registries to satisfy some non-functional aspects of service discovery such as scalability and fault tolerance. The Meteor-S [11] and Pyramid-S [12] systems propose a scalable P2P infrastructure to federate UDDI registries used to publish and discover services; they use an ontological approach to organize registries according to a semantic classification based on the kind of domains that they serve. Even if these solutions address scalability and fault-tolerance, they use a structured topology of registries built on the basis of a domain-specific ontology. This enforces significant constraints on publication policies, hindering a full exploitation of the P2P model.

In the last few years, many P2P systems have been proposed to decentralize the management of information in distributed systems, in particular in Grid and Cloud Computing frameworks. P2P models are classified into *unstructured* and *structured*, based on the way nodes are linked to each other and data about resources is placed on the nodes [13]. In unstructured systems, resources are published by peers without any global planning. This facilitates network management but reduces the efficiency of discovery procedures. In structured systems, resources are associated with specific hosts, often through *Distributed Hash Tables*. However, these systems are not designed to favor the composition of services: the user must first discover the basic services on the network with multiple search procedures, then compose them locally to build the complex application.

To assist the user in the design of composite services, two interesting approaches are based on statistics and process mining. The first technique can be used to help the user identify the services that most likely can complete a composition pattern. The system should be able to recommend a service on the basis of the declared goals and the services already chosen [6]. Process mining aims at the automatic building of composite services starting from the behavior deducible from execution logs of basic services [14]. Process mining is often used when no formal description of the overall process can be obtained by other means, or when the quality of existing documentation is uncertain.

A technique for the spatial clustering and execution of composite services was devised in [15]. In this paper, the authors present an approach to schedule the execution of composite services on the basis of the spatial proximity of files and jobs. The idea is to leave the files at the sites where they have been processed, so that adjacent jobs can immediately retrieve the files and remove them only when they are no longer needed. This approach can only be used in the execution phase, whereas the technique presented here improves the performance at an earlier phase, i.e., the discovery of basic services over the network.

A notable trend for the future of service computing is the autonomic composition of services [4]: the idea is that the system should build composite services by automatically dis-

covering the basic components, choose among the available suppliers and select among the different options available for contracts. Our vision, in which service descriptors are self-organized by agents according to their mutual compatibilities, can be a step towards the autonomic composition of services.

III. An ant-inspired approach for the collective discovery of workflow services

This paper introduces a technique that may greatly facilitate one of the main tasks in the process of building a composite service, or "workflow": the discovery of the basic component services over the network.

Services are generally associated with metadata documents, or "descriptors", which contain the references to the services along with the descriptions of their functionalities. These descriptors are indexed and discovered by means of *keys*, which may be generated by hash functions, as in *Distributed Hash Tables*, or may have a semantic meaning: for example, each bit in the key may correspond to a specific topic, and the 1/0 value may mean that the service covers/does not cover that topic. This strategy is generally adopted in P2P-based distributed systems and service-oriented architectures [16] and will be used in this work. Moreover, to facilitate the management of a wide set of services, they are usually categorized into different classes, according to their semantics and functionalities. The rationale of this classification is that in many distributed environments, such as Grids and Clouds, generally users do not need to discover a single specific service, but collect information about services having specified characteristics or functionalities, for example a mathematical service that provides a numerical solution to differential equations, or a bio-informatic software able to perform particular operations on protein data. A service class is therefore defined as a set of services with common properties. All the services of the same class are assigned the same value of the key, either with a hash function or through a semantic representation in which a bit of the key corresponds to a specific characteristic of the class, as in the example mentioned before. A user can issue a query with a specific target key, and in this way discover a number of services belonging to the corresponding class: then the user will choose those that are the most appropriate for his/her purposes.

In this kind of environment, a user that wishes to build a workflow of services, must first individuate the classes of the component basic services (and the corresponding keys), then issue a search request for each class, and finally select among the services that have been discovered. This process may result in long search times and excessive use of network and computing facilities. Conversely, our approach aims to individuate the descriptors of services that are likely to be co-used in the same workflow, and place them close together. In this way, a single discovery request can locate in a single shot several services that may be used as basic components of a workflow, and it is then possible to reduce the number of discovery requests that are necessary to find all the required services.

Since the class of a service is determined by the descriptor key, in the following we will often refer to a service descriptor by means of its key. The reorganization of keys is achieved through the operations of a number of mobile agents whose behavior is inspired by ant colonies [7] [17]. The agents move the keys over the network and sort them in accordance with their mutual co-use frequencies. This approach is admissible if the co-use of services has non-uniform statistical properties. If a specific class of service is required in a workflow, the probability that another service class is needed in the same workflow is assumed to be not uniform: some types of services are more likely to be needed than others. This assumption is valid in most cases, as the composition patterns of workflows often obey to recurrent patterns [18]. For example, the execution of a data mining software may require a specific algorithm to preprocess the input data: therefore, the preprocessing algorithm and the data mining software will be used together frequently.

The approach presented in this paper uses two different types of mobile agents: the *ant agents* and the *query agents*. This is coherent with the bio-inspired nature of the approach: for example, insect colonies are composed of individuals with different capabilities and can globally optimize the execution of heterogenous tasks via their collective intelligence [7].

These two types of agents are concurrently exploited to perform three distributed algorithms: Alg. 1, for the gathering and dissemination of statistics about the co-use of service classes; Alg. 2, for the reorganization of keys, with the objective of gathering the descriptors of frequently co-used service classes in neighbor peers; Alg. 3, for the efficient execution of discovery procedures.

The three algorithms are concurrently executed on all the peers of the network: the main tasks performed on a single peer are sketched in Figure 1. The figure also depicts the data repositories used by the algorithms: the repository of keys, the list of the adjacent peers, and the *co-use matrix*, which maintains the statistics about the co-use of different service classes in the same workflow. The high level logic of the approach is summarized in the following. When defining the workflow of a complex application, the user individuates the classes of the component services, and generates a *query agent* to find a number of services for each class. The query agent must be driven to the peers that store a considerable number of keys that belong to the specified service classes. The query agent carries the workflow description, with the list of service classes, and the target keys collected in the peers visited so far (see bottom part of Figure 1). At any peer where it is delivered, the query agent performs the following operations: (i) it updates the co-use matrix, which

is exploited by the ant agents to reorganize the keys; (ii) it collects the target keys stored in the local repository; (iii) it hops to the next peer, selected in accordance with the discovery algorithm.

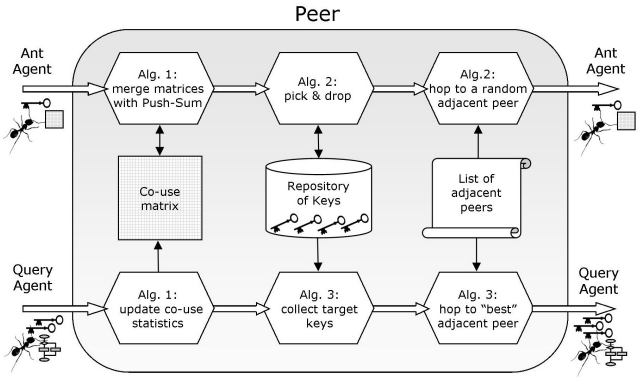

Figure 1. Tasks performed by ant and query agents when they are delivered to a new peer. Each task is associated with one the three basic algorithms.

The ant agents are generated by peers when they join or reconnect to the network, and travel randomly from peer to peer. An ant carries the co-use matrix of the last visited peer and, possibly, one or more keys previously picked on other peers (see the top part of Figure 1). When arriving at a new peer, the ant agent performs the following operations: it merges the carried matrix with the one stored locally, using the *Push-Sum* protocol [19]; it executes pick and drop Bernoulli trials, in order to decide whether or not to drop the carried keys and/or pick other keys stored in this peer; it hops to a new peer, chosen randomly.

It should be remarked here that ant and query agents operate continuously and in parallel. This allows the reorganization of descriptors to adapt dynamically to user requests, so as to serve them more rapidly and efficiently.

The three algorithms are described in the following subsections in detail.

A. Alg. 1: building and dissemination of co-use matrices

The objective of the first algorithm is the building of *co-use matrices* by the peers of the network. We will assume that the services are categorized into N_S classes, according to a domain specific classification or ontology. For convenience, service classes are assigned progressive integer indices from 0 to $N_S - 1$. Each peer maintains a bi-dimensional matrix M, in which the element $M(i, j)$, with $i, j = 0..N_S - 1, i \neq j$, represents the frequency of co-occurrences of service classes i and j in the same workflow.

To build the matrix M, each peer examines the search requests carried by the *query agents* that pass through the peer. Each request lists the service classes the are needed to compose a specific workflow. For each couple of such service classes, i and j, the matrix element $M(i, j)$ is incremented by 1. To give more relevance to the recent behavior of the system, the matrix elements are computed in a temporal window that includes the workflow requests received during the past H hours (H is a tunable parameter, and in this work was set to 24 hours). The matrix values are then normalized, dividing each element by the largest element value, so as to obtain real values between 0 and 1. The co-use matrices, maintained by the different peers, are exploited by ant agents to move and reorganize the service descriptors, as described in the Section III-B.

Information about the co-use of services must be exchanged among the peers, otherwise the matrices stored by different peers would converge very slowly and possibly to different values. The *ant agents*, whose primary duty is the reorganization of the keys, are also assigned the responsibility of disseminating the information contained in the co-use matrices. After hopping from one peer to another, an ant delivers the normalized matrix of the source peer, $\bar{M}_s$, to the target peer (the matrix is symmetric, hence only half matrix must be delivered). The target peer recalculates its co-use matrix $\bar{M}_t$ by combining its current values with the values of the matrix $\bar{M}_s$. This technique follows the well-known *Push-Sum* protocol [19], based on a gossiping strategy, which allows aggregate values, in this case the average values of the $\bar{M}$ elements stored by the different peers, to be computed in a number of rounds that is logarithmic with respect to the number of peers. In this way, the information is progressively disseminated across the network, and the values of the peer matrices can converge rapidly and to uniform values.

B. Alg. 2: reorganization of descriptors

The objective of the second algorithm is the reorganization of descriptors, performed by *ant agents*. An ant agent is generated by a peer when it joins or reconnects to the system, and travels the network, hopping from peer to peer, for a given amount of time. By correlating the lifetime of the ant agent to the average connection time of the issuing peer, it is possible to tune the average number of agents that circulate in the network. For example, if the agent lifetime is set to the average connection time of the peer, it can be easily deduced that the average number of agents is always comparable to the average number of peers connected in the network. Indeed, the agents that die are compensated by the new agents that are generated by peers.

Each ant, at random time intervals, hops from a peer to an adjacent one. At the new peer, the ant tries to *pick* a descriptor that is not frequently co-used with the other descriptors stored in the local region. After picking a descriptor, the ant will carry it and try to *drop* it in a peer where the co-use frequency with the local descriptors is high. The co-use frequency of two descriptors is defined as the value of the element $\bar{M}(i, j)$ of the normalized co-use matrix, where i and j are the classes of the two descriptors.

Since each service class is univocally associated with a specific value of the descriptor key, in the following, for simplicity, the discussion will focus on the reorganization of keys. For example, to say that a descriptor of a given service class is picked/dropped by an agent, we will say that the agent picks/drops a key of that class. Pick and drop operations are driven by corresponding pick and drop probability functions, as described in the following.

Pick operation.

To decide whether or not to pick a key, say key $\widehat{k}$, the agent calculates the "co-use similarity" of this key with the other keys k stored in the local area A, which includes the local peer and the peers directly adjacent to it. The co-use similarity $f(\widehat{k}, A)$ is defined as follows:

$$f(\widehat{k}, A) = \frac{1}{N_k} \cdot \sum_{k \epsilon A} (\overline{M}(k, \widehat{k})) \tag{1}$$

where $\bar{M}$ is the normalized co-use matrix of the local peer (Section III-A), and N_k is the number of keys stored in the local area A. From the definition, the co-use similarity has values between 0 and 1 and a value close to 1 corresponds to a high co-use frequency of $\widehat{k}$ with the other keys of the local area.

The key is picked by the agent, after a Bernoulli try, with probability:

$$P_{pick} = \frac{\alpha_p}{\alpha_p + f(\widehat{k}, A)}, \quad \text{with } 0 \leq \alpha_p \leq 1 \tag{2}$$

Therefore, the pick probability is inversely proportional to the co-use similarity, which ensures that an "outlier" key will be picked with high probability and moved to other regions of the network. The parameter α_p can be tuned to modulate the degree of similarity among keys. In fact, the pick probability is equal to 0.5 when $f(\widehat{k}, A)$ and α_p are comparable, while it approaches 1 when $f(\widehat{k}, A)$ is much lower than α_p (i.e., when $\widehat{k}$ has very low co-use similarity with the other keys) and 0 when $f(\widehat{k}, A)$ is much larger than α_p (i.e., when $\widehat{k}$ has high co-use similarity). In this work, α_p is set to 0.1, as in [7].

Drop operation.

Once a key $\widehat{k}$ is picked, it will be carried by the agent across the network. The agent will evaluate the drop probability function at any new peer, until the key is actually dropped; after that, the agent will try to pick another key. The drop operation also depends on a Bernoulli try, whose probability is:

$$P_{drop} = \frac{f(\widehat{k}, A)}{\alpha_d + f(\widehat{k}, A)}, \quad \text{with } 0 \leq \alpha_d \leq 1 \tag{3}$$

In this case the co-use similarity $f(\widehat{k}, A)$ is evaluated between the carried key, $\widehat{k}$, and the keys stored in the new local area A. Notice that the drop probability is directly proportional to the co-use similarity. The value of α_d should be higher than α_p, in order to moderate the drop probability and give the agent the possibility of bringing the key to a remote area, if needed. In this work, α_d is set to 0.5.

Pick and drop operations are used to cluster keys of service classes having high co-use frequencies, and avert keys of rarely co-used service classes. The efficiency of this technique lies in the "swarm intelligence" behavior of ant-inspired algorithms. Each single ant performs very simple operations without any knowledge about the overall process, but the combined operations of several agents lead to the emergence of an organized and flexible information system, in which the keys are spatially sorted according to the adopted definition of similarity. Being completely decentralized, the key reorganization process adapts easily to the modifications of the environment, for example to the disconnections/reconnections of peers and to changes in the co-use frequencies. The ant algorithm is also robust with respect to the values of the parameters α_p and α_d: these values can have an impact on the velocity and duration of the transient phase of the sorting process (i.e., the phase in which the keys are reorganized starting from complete disorder), but they have little influence on the performance observed under steady conditions (in which the keys are already sorted, and agent operations have only to cope with environmental modifications, such as peer disconnections, publication of new keys, etc.).

C. Alg. 3: collective discovery of services

In the design phase, the user individuates the classes of services that are needed to build a workflow. The purpose of the discovery process is to find as many services as possible that belong to a set of target classes. The reorganization of keys explained in the previous section is exploited to increase the efficiency of the discovery process.

When a peer initiates a discovery process, it issues a query and consigns a list of target key values (each corresponding to a service class) to a *query agent*. The query agent operates as follows:

i) first, it selects the "closest" service class, i.e., the service class for which it is most likely to find a significative number of keys in the neighbor peers. This is done on the basis of the distance metric defined by the co-use similarity. Specifically, the agent calculates the similarity function of each workflow target key with the keys stored in the local area, using expression (1), and selects the key value k_x, and the corresponding service class, that maximizes this function. The objective is to find as many keys as possible with the chosen value k_x.

ii) the agent hops to the neighbor peer in which the similarity function of the key value k_x is the largest. Owing to the spatial sorting of the keys, each hop allows the query agent to approach a region of the network in which a relevant

number of keys having the value k_x can be found. This step is repeated until it is no longer possible to improve the similarity function, which means that the desired region has been reached (or, in some unfortunate cases, that the query has reached a local maximum). Now, the query agent removes k_x from the set of workflow key values that must still be selected.

iii) the agent turns to step i) and selects the next key value k_y that maximizes the similarity function, among those contained in the query and not yet selected. The agent executes step ii) to discover keys with the new target value k_y.

The procedure terminates when all the key values of the workflow list have been selected. During the search path, the query agent collects all the keys whose value is one of those specified in the query. At the end of the procedure, the query agent goes back to the requesting peer, which examines the keys discovered for each service class. If the number and quality of the results is satisfactory, the discovery process terminates. Otherwise, a new query agent is issued to search for the service classes for which the required number of keys has not yet been discovered. The whole process terminates when the required number of keys has been found for each service class, or when it is not possible to discover more keys.

As an example, let us assume that the workflow must be composed of three services belonging to the classes C_1, C_2 and C_3. A query agent is issued in which the corresponding key values are specified, say k_1, k_2 and k_3. The query agent is successively driven towards the keys having each of these values, and the order is determined by the similarity function, as described before. Suppose that, at the end of the procedure, the query agent returns with 10 keys having the value k_1, 6 having the value k_2, and 12 having the value k_3. If the required number of keys per class was set to 10, the request has been satisfied for classes C_1 and C_3, but not for C_2. Therefore, another query agent is issued to find at least 4 keys with value k_2. If the second agent succeeds, the whole request has been satisfied, with two search procedures, and two query agents.

Notice that, in absence of key reorganization, a different query should be issued for each service class of the workflow. Conversely, owing to the described reorganization of keys, it is possible to reduce the number of queries, down to a single query in the most fortunate cases. This aspect will be examined in Section IV-D.

IV. Performance evaluation

The performance of the presented framework was evaluated with regard to two aspects: the capacity of reorganizing the descriptor keys and the effectiveness of discovery operations. The computational load and network traffic were also assessed. Accordingly, three sets of performance indices were considered:

Indices concerning key reorganization:

- *Peer homogeneity*
  The objective of the ant agents is to place the keys of frequently co-used services close to each other. The *homogeneity* of a peer is defined as the average value of the co-use matrix elements, evaluated for every couple of keys stored in the peer:

$$O_m(P) = \frac{1}{N_{k_1,k_2}} \cdot \sum_{k_1,k_2 in P} (\overline{M}(k_1,k_2)) \qquad (4)$$

  $\overline{M}$ is the normalized co-use matrix of peer P and N_{k_1,k_2} is the number of key couples. To have an overall perspective about the network condition, the index is averaged over all the N_p peers of the network: $O_m = 1/N_p \cdot \sum_P (O_m(P))$
- *Similarity of a peer with its neighbors*
  While the homogeneity index gives information about the keys maintained by a single peer, the discovery procedure also exploits the fact the keys are spatially ordered over the network. This ordering is assessed through the *similarity* index of a peer, defined as the average value of the $\overline{M}$ elements, evaluated for every couple in which the first key is stored in the local peer and the second is located in one of the other peers of the local region A:

$$S_n(P) = \frac{1}{N_{k_a,k_b}} \cdot \sum_{k_a in P, k_b in \{A-P\}} (\overline{M}(k_a,k_b)) \qquad (5)$$

  Again, the value is averaged over the whole network, $S_n = 1/N_p \cdot \sum_P (S_n(P))$. A high value of this index means that the keys stored in adjacent peers are similar and are spatially sorted.

Indices concerning the discovery process:

- *Percentage of "satisfied" service classes*
  Defined as the percentage of service classes, specified in the discovery request, for which a minimum number of keys are discovered. This minimum number, referred to as T_q, is a threshold that can be used as a parameter.
- *Number of discovered keys*
  Average number of discovered keys per each service class specified in the workflow request.

Indices concerning traffic and processing load:

while the previous indices measure the efficacy of the discovery process, the next two indices assess its efficiency and its impact on the network load. The last index measures the impact of the ant algorithm.

- *Number of queries needed to satisfy a discovery request*
  Average number of query agents that are necessary to collect at least T_q keys per each service class specified in the workflow request.
- *Number of steps performed by query agents*

Average number of steps performed by the query agents issued during a workflow discovery process.

- *Frequency of ant agents received by a peer*
  This index is useful to assess both the network traffic and the processing load, since the reception of an ant triggers a computation procedure on the local peer (evaluation and possible execution of a pick/drop operation, recalculation of the co-use matrix with the Push-Sum protocol).

A. Description of the scenario

The performance indices were evaluated in a sample scenario, with 2,500 peers connected in a scale-free topology, in which the number of connections of a peer follows the power-law distribution [20], and the average is set to 4 neighbors per peer. Each peer publishes 15 services on average, with the actual number extracted from a Gamma statistic distribution. The class of each service is generated randomly, with a uniform distribution, among one of the N_S service classes defined by the domain classification or ontology.

The algorithms are evaluated in a scenario in which the co-use of services is not uniform: some couples of service classes are often used together in the same workflow, other couples are co-used more rarely. Unfortunately, this kind of non-uniformity is difficult to predict and is subject to changes: co-use statistics depend on the categorization of services and on the behavior of users in different contexts. To test a realistic scenario, we chose to exploit the classification of services that is provided by semantic tools and ontologies in different application domains. This approach is increasingly popular in the world of service-oriented systems [21]. The most common classification scheme, though not the only one, is the hierarchical categorization of service classes, by means of tree structures: the classification is refined at each level of the tree, and each class corresponds to a leaf node. This kind of classification is adopted, as an example, by the MS-Analyzer platform [22]. MS-Analyzer categorizes the services used to preprocess and mine data produced by mass spectrometry tools, which measure the molecular mass of biological samples. The similarity between two service classes can be measured by their "kinship" degree, that is, by the distance from the closest common ancestor in the tree: the lower is this distance, the higher is the similarity. In this context, it is assumed here that the probability of using two service classes in the same workflow increases with their similarity. This is reasonable, since a service workflow is designed to devise a solution in a specific domain, and it is very likely that the adopted services are categorized within the same domain.

To test this scenario, we made the following specific assumptions, which are coherent with the MS-Analyzer example cited before:

- services are categorized according to a tree structure having l levels and a maximum of m children per node. In our experiments, l is set to 4 (root included), m is set to 3, and the tree is complete, so that N_s=27 different classes of services are defined;
- each workflow is composed of a variable number of services, with a Zipf distribution around the average. In our experiments the average number of services is set to 4: it means that 4-services workflow are the most frequent, followed by workflows with 3 or 5 services, and so on;
- the first service class of a workflow is chosen randomly, with uniform distribution. Each following service class is chosen as follows: first, one of the already chosen classes is selected randomly, as a reference class; then, the new service class is chosen by randomly generating the kinship degree between the reference and the new class. The kinship is generated with a geometric distribution, so that a strict degree of kinship is more probable than a looser one. For example, it is highly probable that the distance from the closer ancestor is 1 (i.e., the reference and the new service class are siblings), while there are lower probabilities that this distance is 2 or 3. Finally, the new service class is randomly chosen among all the classes that have the specified degree of kinship with the reference class.

These assumptions define a specific sample scenario in which our framework was tested. However, it should be remarked that the framework does not depend on any particular scenario: the only necessary assumption is that the co-use statistics of service classes have some form of non-uniformity, which is exploited to reorganize the keys and lead the query agents towards their targets.

The simulation experiments were performed with an event-based simulator similar to that described in [17]. Users issue collective discovery requests at the rate of one request issued from a single peer every 1000 seconds. The service classes specified in the requests are generated as detailed before.

Each new or reconnecting peer generates a new ant agent, whose lifetime is set to the average connection time of the peer. In turn, the average connection time of a single peer is extracted from a Gamma distribution whose average is the average connection time of all peers. This global average is set to 4 hours. Such a setting ensures that the overall number of agents is kept approximately constant, despite the peer disconnections and reconnections, and the turnover of agents. In particular, the number of circulating ant agents is always comparable to the number of peers connected to the network. Finally, the time interval between two successive hops of an ant agent, referred to as T_{mov}, is set to 60 seconds.

B. Performance of the key reorganization process

The homogeneity and similarity indices, averaged over all the peers of the network, are used to verify that the keys of frequently co-used services are actually put in neighbor peers. In Figure 2, these indices are plotted versus time, starting from the instant in which the process is kicked off and the agents begin to travel the network. The homogeneity and similarity indices increase rapidly in the first phase and then get stabilized at values of around 0.77 and 0.62, respectively. The homogeneity index is higher because it is evaluated on the keys stored on a single peer. The trend of both indices, however, proves that keys of co-used services are stored in the same or in neighbor peers, and that they are spatially sorted, which is the condition that allows the discovery algorithm to be executed efficiently. It should be noticed that the transient phase occurs only once, when the process is initiated in a completely disordered system. After this phase, the indices are stable because every change in the network (connections and disconnections of peers, publishing and removals of resources) is easily and rapidly tackled by the agents, thanks to the robustness and flexibility characteristics of the ant algorithm.

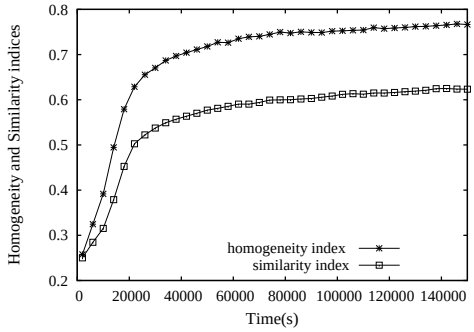

Figure 2. Values of homogeneity and similarity indices vs. time, starting from a disordered network.

C. Performance of resource discovery

The reorganization of keys ensures that the descriptors of the service classes specified in a workflow request are very likely to be stored in neighbor peers: this fact is exploited by the discovery process, as explained in Section III-C. Figure 3 reports the percentage of service classes, specified in a search request, for which the number of keys discovered by the query agents exceeds the threshold T_q, which is varied. The dashed curves correspond to the percentage obtained with the first query agent: it can be noticed that even a single query is sufficient, in steady conditions, to retrieve the desired number of keys for a percentage of service classes between 68% and 82%. Of course the percentage decreases as the threshold value T_q increases. As discussed in Section III-C, new query agents are issued to satisfy the remaining service classes, until all of them have been satisfied or it proves impossible to fulfill the goal (a new query agent does not discover any further key). At the end of the process the percentage of satisfied service classes is practically equal to 100%, as shown by the continuous curves in Figure 3.

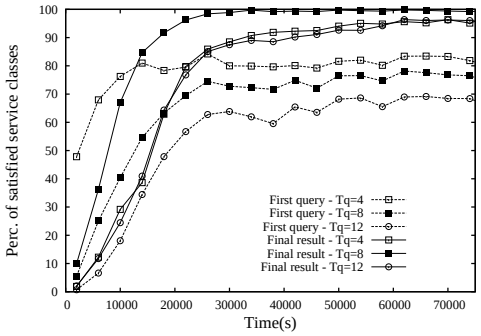

Figure 3. Perc. of service classes of a workflow for which at least T_q keys are discovered after the first query (dashed lines) and at the end of the discovery process (continuous lines), for different values of the threshold T_q.

Figure 4 shows that the average number of keys discovered per service class increases with time and converges to around 25 keys, which is a much larger value than the requested minimum, the threshold T_q. The rationale of this increase is that the discovery algorithms can direct the query agents to regions where many useful keys can be found.

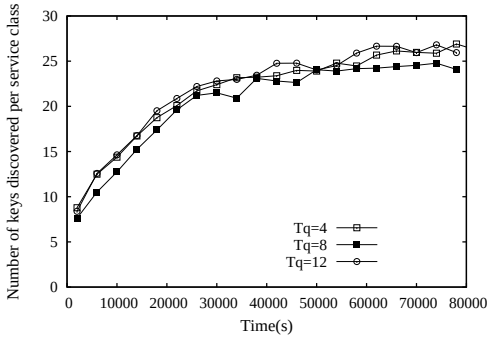

Figure 4. Average number of keys discovered per service class. The index is reported for different values of the threshold T_q.

D. Traffic and load analysis

The average number of query agents that are necessary to obtain the required number of keys, for all the service classes specified in the request, is depicted in Figure 5. In steady conditions, this number is always lower than 1.4. Notice that, with no key reorganization, about 4 queries would be necessary, because the average number of services that compose a workflow was set to 4. This means that the number of queries can be reduced on average by 65%.

Figure 6 reports the total number of steps that are performed by all the query agents of a discovery process. This number decreases with time, which proves that the reorganization of keys not only improves the effectiveness

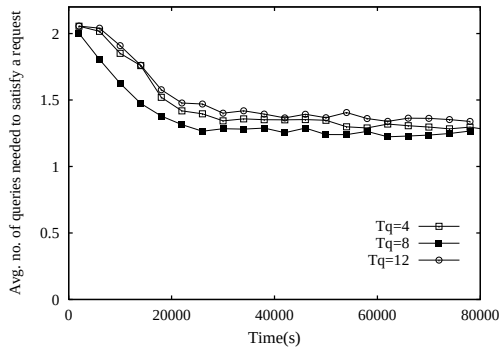

Figure 5. Average number of query agents needed to complete the discovery process, for different values of the threshold T_q.

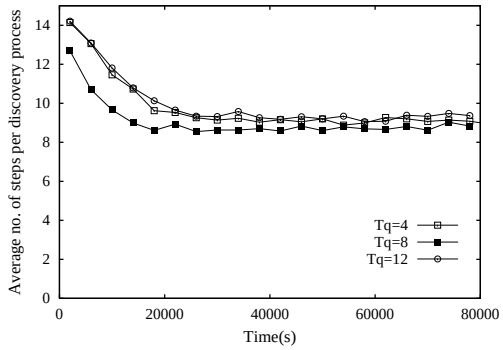

Figure 6. Average number of steps performed by query agents, for different values of the threshold T_q.

of the process, but also shortens the average path of query agents.

The number of ant agents per second that arrive and are processed at a peer, referred to as L, can be calculated by multiplying the average number of ants $\overline{N}_a$ by the frequency of their movements $1/T_{mov}$, so obtaining the number of times per second that an ant arrives at any peer, and then dividing the result by the average number of peers $\overline{N}_p$ to get the number of times per second that an ant arrives at a specific peer,

$$L = \frac{\overline{N}_a}{\overline{N}_p \cdot T_{mov}} \tag{6}$$

In the considered scenario, $\overline{N}_a$ and $\overline{N}_p$ are comparable, as discussed in Section IV-A, therefore the average frequency of received ants is equal to the inverse of T_{mov}, i.e., one ant every 60 seconds. This result was confirmed by the simulation experiments. Since the operations performed after receiving an ant are very simple and fast, this can be considered an acceptable load.

Finally, it should be noted that the performance indices do not depend on the network size, which was confirmed by simulation experiments. This scalable behavior descends from the decentralized and self-organizing nature of the adopted algorithms, and is a typical advantage of the swarm intelligence paradigm.

V. Conclusions

This paper aims to make a novel contribution to the design and development of a service-oriented framework that assists the user in the building and execution of composed services. The basic idea is to place the descriptors of services close together if they are often co-used in workflows, and exploit this reorganization to facilitate the concurrent discovery of the basic services that will compose a new workflow. The devised algorithms, partly inspired by the behavior of ant colonies, are completely decentralized and self-organizing, which avoids the presence of bottlenecks, and guarantees good fault-tolerance and adaptivity properties. The evaluation of the algorithms, performed through simulation experiments, confirmed the efficiency of the information reorganization process and the effectiveness of the collective discovery procedures.

Acknowledgment

This work was partially funded by the MIUR project DM21301, "OpenKnowTech: Laboratory of Technologies for the Integration, Management and Distribution of Data, Processes and Knowledge".

References

[1] European Community for Software & Software Services, "White paper on software and service architectures, infrastructures and engineering - action paper on the area for the future eu competitiveness, Tech. Rep. available at http://www.eu-ecss.eu/documentation/ecss-documentation, January 2009.

[2] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[3] B. Hayes, "Cloud computing," *Communications of the ACM*, no. 7, pp. 9–11, July 2009.

[4] S. Dustdar and M. P. Papazoglou, "Services and service composition - an introduction," *Information Technology*, vol. 50, no. 2, pp. 86–92, 2008.

[5] B. Srivastava and J. Koehler, "Web service composition - current solutions and open problems," in *Proc. of the ICAPS 2003 Workshop on Planning for Web Services*, Trento, Italy, June 2003, pp. 28–35.

[6] P. Wisner, "Automatic composition in service browsing environments," in *MIRW 2006, Workshop on Mobile Interaction with the Real World*, Espoo, Finland, September 2006.

[7] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm intelligence: from natural to artificial systems*. New York, NY, USA: Oxford University Press, 1999.

[8] J. Rao and X. Su, "A survey of automated web service composition methods," in *Workshop on Semantic Web Services and Web Process Composition, SWSWPC, Springer LNCS, Vol. 3387*, San Diego, CA, USA, July 2004, pp. 43–54.

[9] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[10] E. Zimeo, A. Troisi, H. Papadakis, P. Fragopoulou, A. Forestiero, and C. Mastroianni, "Cooperative self-composition and discovery of grid services in P2P networks," *Parallel Processing Letters*, vol. 18, no. 3, pp. 329–346, September 2008.

[11] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller, "METEOR-S WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of web services," *Information Technology and Management*, no. 1, pp. 17–39, January 2005.

[12] T. Pilioura, G.-D. Kapos, and A. Tsalgatidou, "PYRAMID-S: A scalable infrastructure for semantic web service publication and discovery," in *RIDE '04: Proc. of the 14th International Workshop on Research Issues on Data Engineering*, Boston, MA, USA, March 2004.

[13] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys*, vol. 36, no. 4, pp. 335–371, 2004.

[14] W. Van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[15] L. Meyer, J. Annis, M. Wilde, M. Mattoso, and I. Foster, "Planning spatial workflows to optimize grid performance," in *SAC'06: Proc. of the 2006 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2006, pp. 786–790.

[16] I. J. Taylor, *From P2P to Web Services and Grids: Peers in a Client/Server World*. Springer, 2004.

[17] A. Forestiero, C. Mastroianni, and G. Spezzano, "So-grid: A self-organizing grid featuring bio-inspired algorithms," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 3, no. 2, May 2008.

[18] H. Meng, L. Wu, T. Zhang, G. Chen, and D. Li, "Mining frequent composite service patterns," in *GCC '08: Proc. of the 2008 Seventh International Conference on Grid and Cooperative Computing*, October 2008.

[19] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *Proc. of the 44th Annual IEEE Symposium on Foundations of Computer Science*, 2003.

[20] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, October 1999.

[21] T.-F. Fortis and C. Mindruta, "Ontologies in a service oriented computing environment," in *Proc. of the 4th International Workshop on Workflow Management ICWM 2009*, Geneva, Switzerland, May 2009.

[22] M. Cannataro and P. Veltri, "Ms-analyzer: preprocessing and data mining services for proteomics applications on the grid," *Concurrency and Computation: Practice & Experience*, vol. 19, no. 15, pp. 2047–2066, 2007.